МУНИЦИПАЛЬНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ПЫЩУГСКАЯ СРЕДНЯЯ ОБЩЕОБРАЗОВАТЕЛЬНАЯ ШКОЛА

МЕТОДИЧЕСКОЕ ПОСОБИЕ ДЛЯ УЧИТЕЛЯ АНАЛИЗ ЗАДАЧ ПРОГРАММИРОВАНИЯ НА ЕГЭ ПО ИНФОРМАТИКЕ

Вагина Евгения Николаевна
учитель информатики и ИКТ

2019год

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Цель моего пособия: разработка и теоретическое обоснование методики подготовки учащихся к решению задач программирования в ЕГЭ по информатике. Не секрет, что многие учителя информатики – совместители, основная специальность которых биология, математика, начальные классы и т.п.. Сама являясь учителем математики по основной специальности, знаю, насколько не профессионалу трудно научиться программированию и научить этому ребят. Но сейчас, имея двадцатилетний стаж учителя информатики, программирование считаю самой интересной, увлекательной темой в информатике, способной увлечь большую часть думающих учащихся, определить их профессиональный выбор. Возможно, мои наработки помогут учителю информатики в его нелёгком труде по подготовке учеников к государственному экзамену по информатике за курс средней школы.

Данная методика подготовки к ЕГЭ позволит повысить:

- интерес к изучению информатики и программированию
- уровень усвоения основ алгоритмизации и программирования, соответствующих стандарту среднего общего, базового и профильного уровней образования;
- уровень усвоения основных элементов содержания, проверяемых на ЕГЭ

Для достижения цели исследования решались следующие задачи:

- Изучались основные тенденции развития современного школьного курса «Информатика и ИКТ»: цели, содержание, формы, методы и средства обучения;
- Перечень элементов содержания, включенных в кодификатор и проверяемых на ЕГЭ;
- Состояние методической системы подготовки учащихся к ЕГЭ по информатике;
- Трудности, возникающие у учащихся при решении задач на алгоритмизацию и программирование.

Для решения поставленных задач мной были использованы следующие методы исследования:

- Изучение и анализ научно-методической, психолого-педагогической, учебной и специальной литературы по проблеме исследования;
- Изучение и анализ научно-методической, учебной и специальной литературы по подготовке к ЕГЭ;
- Изучение и анализ учебно-методической документации (учебных программ, планов, нормативных документов, методических руководств).
- Изучение и анализ перечня элементов содержания, проверяемых на ЕГЭ
- Анализ результатов решения задач программирования на ЕГЭ по информатике за 2015 – 2018 годы.

Для учителя информатики подготовка учащихся к ЕГЭ по информатике и ИКТ носит довольно широкий, разноплановый характер. Это обусловлено и различным уровнем подготовки учащихся, и различными условиями изучения непосредственно самого предмета в образовательном учреждении (количество часов, обеспечение техникой, квалификацией самого учителя и т.д.).

Вообще ситуация непростая: учитель информатики в современной школе зажат в тиски: с одной стороны ЕГЭ, с другой стороны образовательная программа, с третьей -

социальный заказ общества, который требует от учащихся конкретных практических навыков работы на компьютере и умение анализировать программы, знать языки программирования.

А часов на изучение информатики в непрофильной школе очень мало – всего по 1 часу в 7 – 8 классах и 2 часа в 9 классе, и по 1 часу информатики в 10 и 11 классах. Перед учителем информатики стоит сложная задача. С одной стороны, учащимся надо дать такие знания, чтобы они смогли успешно подготовиться к выбранной профессиональной деятельности, продолжать образование в течение всей жизни, жить и трудиться в условиях информационного общества. С другой стороны, нужно подготовить учащихся к ЕГЭ, главной целью введения которого является получение объективной оценки качества подготовки выпускников средней школы.

Назначение экзаменационной работы не только оценить общеобразовательную подготовку по информатике и ИКТ выпускников XI классов общеобразовательных учреждений, но и конкурсный отбор абитуриентов в учреждения среднего и высшего профессионального образования.

Наша школа, как и большинство сельских школ не является профильной. Но наши ученики выбирают информатику на ЕГЭ, планируют и поступают в Вузы, где информатика как профильный вступительный экзамен указана в большом числе специальностей: «Прикладная математика и информатика», «Информационные технологии», «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника», «Робототехника», «Информационные технологии в медиаиндустрии», «Вычислительные машины, комплексы, системы и сети», и т. д..Ученики сельской школы тоже вправе достойно сдать ЕГЭ, и учитель обязан им в этом помочь. Одной из составляющих успешности учителя является успех его учеников. Подготовка к ЕГЭ – это всегда ответственный процесс. И от того, насколько грамотно построен будет этот процесс, зависит наш результат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Внедрение государственной итоговой аттестации ЕГЭ в России сделало существенный шаг навстречу повышению объективности и унификации контроля результатов обучения. Основным назначением **ЕГЭ (единого государственного экзамена)** – **оценить общеобразовательную подготовку по информатике** и ИКТ выпускников XI (XII) классов общеобразовательных учреждений с целью проведения итоговой аттестации выпускников общеобразовательных учреждений и конкурсного отбора абитуриентов в учреждения среднего и высшего профессионального образования. ЕГЭ проводится в соответствии с Федеральным законом от 29.12.2012 № 273-ФЗ «Об образовании в Российской Федерации

Документы, определяющие содержание КИМ

Содержание экзаменационной работы определяется на основе приказа Министерства образования РФ от 05.03.2004 № 1089 «Об утверждении Федерального компонента государственных образовательных стандартов начального общего, основного общего и среднего (полного) общего образования».

Подходы к отбору содержания, разработке структуры КИМ

В государственном стандарте по информатике отмечается, что в результате изучения информатики и ИКТ на базовом уровне ученик в области программирования должен:

1. знать основные свойства алгоритмов, типы алгоритмических конструкций: следование, ветвление, цикл, понятие вспомогательного алгоритма;
2. уметь использовать алгоритмические конструкции, выполнять и строить простые алгоритмы, выполнять базовые операции над объектами: цепочками символов, числами, списками, деревьями;
3. использовать приобретенные знания и умения в практической деятельности и повседневной жизни при выполнении индивидуальных и коллективных проектов, в учебной деятельности, в дальнейшем освоении профессий.

Данные знания, умения и навыки формируются при изучении темы «Алгоритмизация и программирование».

Проанализируем государственные нормативные документы: Федеральный базисный учебный план для образовательных учреждений РФ отводит 105 часов для обязательного изучения информатики и информационных технологий на ступени основного общего образования и 70 часов на ступени полного общего образования. На алгоритмизацию и программирование в 10 классе отводится около 19 часов. В случае профильного изучения информатики ситуация значительно улучшается. Федеральный базисный учебный план отводит 280 часов для обязательного изучения информатики и информационных технологий на ступени среднего общего образования, то есть по 140 учебных часов на каждый год обучения. Как мы видим, на изучение раздела программирования в непрофильных классах отводиться недостаточное количество времени, это приводит к тому, что изучение некоторых тем проходит поверхностно, а некоторые исключаются вовсе.

Следует обратить внимание и на затруднение в освоении алгоритмизации и программирования у значительной части учащихся. Данная тема, и с точки зрения учителей, и с точки зрения учеников, является сложнейшей в рамках учебного предмета. К сожалению, необходимо отметить и низкий уровень квалификации многих учителей информатики (часто это учителя – совместители), что не способствует качественному освоению предмета.

С моей точки зрения, учитывая приведенные выше факторы, представляется актуальной задача совершенствования методики преподавания программирования в средней школе.

Важную роль в методике обучения программированию, следует отводить самостоятельной работе учеников, так как только самостоятельная разработка алгоритмов и программ, должным образом способствует развитию алгоритмического мышлению и закреплению необходимых навыков. Программирование — нелегкая задача. Она требует от вас концентрации, особенно когда вы изучаете новые вещи. Это тяжело для мозга, так что случаются моменты, когда вы не понимаете ни почему код не работает, ни, тем хуже, почему он вдруг заработал сразу после написания. Это действительно сложно.

Полагаю, в чистом виде программирование интересует небольшую категорию людей. Теория алгоритмов или программирование - это чересчур специальные вещи на сегодняшний день, когда компьютеры продаются в супермаркетах рядом с телевизорами и DVD-проигрывателями. Сегодня простому пользователю программировать не нужно, хотя еще недавно такого просто не могло быть. Поэтому достаточно часто слышишь вопрос: Зачем всех подряд учить программированию, если это реально нужно нескольким ученикам собравшимся в технический вуз причем на соответствующие специальности? Большинство людей, использующих компьютеры, не пишут своих собственных программ, и им практически вообще не требуется знать программирование. Если рассуждать, что пользователю достаточно знать только "три кнопки", и на информатике в школе нужно давать только пользовательский курс, то по аналогии можно утверждать, что на математике нужно учить пользоваться калькулятором, зачем школьникам эти логарифмы, производные, интегралы, если есть компьютер, калькулятор, да и таблицу умножения знать не нужно, главное уметь кнопки на элементарном уровне нажимать.

Однако я уверена, что изучать программирование обязательно нужно! Изучая программирование, ученики лучше понимают сущность работы компьютера, его возможности и ограничения. Программирование помогает школьникам развивать навыки мышления, а также привычку к аккуратной работе. Нет лучшего способа развить логику мышления, точность формулировок, аккуратность, чем программирование. Ряд школьных предметов вообще не связан с какой-либо стороной мышления, а настроен на усложнение знаний в конкретной области, на развитие кругозора учащихся. Информатика развивает специфический стиль мышления.

Программирование – это такая основополагающая вещь которую хоть в малой степени, но надо знать всем. Считаю, что умение строить алгоритмы и программировать их на алгоритмических языках отлично развивает логическое мышление.

Мне кажется, что если человек понимает как создаются программные средства и умеет хотя бы на элементарном уровне программировать, то он лучше будет понимать принцип работы любого прикладного ПО, и в случае ошибки или в случае нестандартной ситуации будет знать что делать и сможет справиться с ней.

При построении обучения учащихся теме «Алгоритмизация и программирование» каждый учитель информатики сталкивается с огромным количеством вопросов: как построить изложение материала, какие использовать методические разработки, в какой форме проводить занятия, какие составить практические задания, какой материал использовать учащимся при изучении и другие. Все эти вопросы возникают из-за отсутствия четко и в полном объеме изложенных учебно-методических материалов и учебников для изучения данной темы.

На учебный предмет «Информатика и ИКТ» в федеральном базисном учебном плане в 8-х и 9-х классах отводится 105 часов (35 учебных часов из расчета 1 учебный час в неделю в 8 классе и 70 учебных часов из расчета 2 учебных часа в неделю в 9 классе). Из этого количества часов отводится 19 часов на изучение темы «Алгоритмы и исполнители», причем подразумевается изучение формальных исполнителей алгоритмов. Среднее (полное) общее образование базового уровня включает в себя 35 часов в 10 классе и 35 часов в 11 классе (из расчета 1 учебный час в неделю). В данное количество часов входят 19 часов на изучение темы «Алгоритмизация и программирование».

Также предполагается, что учитель будет использовать язык программирования во время решения задач при изучении других тем.

Таким образом, объём часов на изучение темы «Алгоритмизация и программирование» не дает возможности в полной мере изучить данную тему в школьном курсе. В этом и заключается несоответствие выделяемого количества часов на изучение данной темы с объемом рассматриваемого материала за данное количество часов, и в этом выражается несоответствие к требованиям выпускника по форме единого государственного экзамена.

И вот тут и проблема как заинтересовать учеников программированием, как научить понимать и решать задачи. Как заставить их самим заниматься этим увлекательным делом. Из опыта работы каждый из учителей информатики (в том числе и я) может сказать, что наибольший интерес у учащихся вызывает графика (как в восьмых, так и в одиннадцатых классах), при работе с которой на экране виден красочный результат выполнения программы. Мы знаем, как порой трудно объяснить учащимся, что все, что выполняет компьютер – это программы. Поэтому у меня возникла идея давать большую часть материала, используя графику. Отсюда вытекает и изменение учебного плана занятий: на первое место можно сразу поставить изучение графических операторов, а затем уже с их помощью объяснять (по возможности) весь остальной материал. Ребенку проще увидеть и сделать, чем пытаться понять, что так происходит на самом деле. Используя подобные программы, можно проиллюстрировать работу всех основных операторов, а если использовать элементы блок-схем, то и продемонстрировать работу алгоритмов без использования операторов. Также хороший отклик у ребят особенно 5 – 8 классов находит использование на уроках исполнителей типа Робот, Чертежник и т.д. И в PascalABC обязательно показываю использование модуля GraphABC Практически все ученики в классе заинтересовываются в решении таких задач. Затем самостоятельно решают уже более сложные задачи.

Применяя такой опыт в работе, можно наблюдать у учащихся некий элемент соревновательности, желание сделать лучше и красивее своего товарища. Данный тип уроков приносит результаты как в младших (учитывая их возраст), так и в старших классах.

Учащиеся для успешной сдачи экзамена должны не только знать основные алгоритмические конструкции и операторы изучаемого языка программирования, но и иметь опыт самостоятельной записи алгоритмов и программ, решения практических задач методом разработки и отладки компьютерной программы. Следует уделять больше внимания формализации записи и исполнения алгоритмов, так как многолетний опыт показывает, что у части учащихся так и не формируется умение формального исполнения алгоритмов.

Школьная информатика в России начиналась с алгоритмизации и программирования, как с основной темы курса. В то время даже был провозглашен лозунг: "Программирование - это вторая грамотность". Эта тема изучалась и в безмашинном варианте, и с компьютерной поддержкой на БК и Yamaha - компьютерах, первыми появившимися в школах. Основным программным обеспечением данных компьютеров был встроенный язык программирования Бейсик. Но последние годы характеризовались уменьшением количества часов на изучение алгоритмизации и программирования в старшей школе, что было связано с развитием школьной информатики как самостоятельного предмета и бурным развитием информационных технологий. Чрезмерное увлечение «пользовательской компонентой» вытеснило изучение этих вопросов не только из некоторых профильных курсов, но даже из ряда учебников базового курса. При явном улучшении оснащения школ компьютерной техникой уровень общеобразовательной подготовки выпускников заметно снизился.

Похожие проблемы появились во многих странах. Например, несмотря на перенасыщенность американских школ компьютерной техникой, на всеобщую доступность

компьютеров и сети Интернет, нет положительных сдвигов в уровне общей подготовки учащихся. Полное отсутствие представлений об алгоритмизации и технологиях программирования у выпускников школ вызывает беспокойство преподавателей колледжей и университетов и приводит к изменению учебных планов в сторону продолжительности обучения на вводных курсах. По мнению зарубежных ученых и специалистов в области образования вопросы, связанные с алгоритмизацией и программированием являются фундаментальными и обязательно должны изучаться на вводных курсах информатики вне зависимости от дальнейшего профиля обучения.

При изучении содержательной линии «Алгоритмизация и программирование» следует рассматривать три аспекта: теоретический, развивающий и прагматический.

Чтобы разобраться в этих вопросах, разделим содержательную линию на два предмета обучения: обучение алгоритмизации и обучение программированию на ЭВМ (языкам программирования).

Цель обучения алгоритмизации заключается в овладении учащимися структурной методикой построения алгоритмов. Это значит, ученики должны научиться использовать в практике построения алгоритмов основные управляющие структуры: следование, ветвление, цикл; уметь разбивать задачу на подзадачи, применять метод последовательной детализации алгоритма. Дидактические средства для этого хорошо отработаны - это разнообразные учебные исполнители алгоритмов: черепахи, роботы, чертежники и пр. Известна методическая идея, идущая еще от А.П.Ершова: исполнители алгоритмов делятся на исполнителей, работающих «в обстановке» и исполнителей, работающих с величинами. Перечисленные выше исполнители относятся к первой группе.

Использование таких исполнителей с методической точки зрения очень эффективно. Основные достоинства - понятность решаемых задач, наглядность работы исполнителя, поддержка структурной методики алгоритмизации. Задача развития структурного алгоритмического мышления учащихся решается в полной мере на учебных исполнителях, работающих «в обстановке».

При изучении алгоритмизации в пропедевтическом курсе развивающий аспект является основным. Однако в базовом курсе информатики к нему добавляются еще новые аспекты, которые следует отнести к теоретическим аспектам. Таких аспектов два. Первый - кибернетический аспект. Речь идет о знакомстве с информационными основами процессов управления. Место алгоритмов в этой теме определяется следующим тезисом: алгоритм управления - это информационная составляющая всякой системы управления. В процессе управления происходит передача данных о состоянии управляемого объекта по линии обратной связи, а по линии прямой связи - управляющая информация, т.е. команды управления. Последовательность команд управления и составляет алгоритм управления. Его должен «знать» управляющий объект. Учебные исполнители алгоритмов являются прекрасными моделями процессов управления. На них, в частности, хорошо иллюстрируется тот факт, что без обратной связи алгоритм управления может быть только линейным, а при наличии обратной связи может содержать ветвления и циклы.

Радует, что появляются хорошие учебники, где программированию уделяется должное внимание. Это и базовый курс И. Г. Семакина 10 класс, и углубленный курс К. Ю. Полякова, 10 – 11 класс, учебники Л.Л.Босовой 8,9 и 11 класс.Всегда пользуюсь прекрасной книгой В. И.ТишинаПрограммирование на Паскале. Практикум. При подготовке учащихся к ЕГЭ надо обращать внимание прежде всего на темы, включенные в программы для поступающих в вузы: алгоритмизацию и программирование.

ТЕПЕРЬ О ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Наша школа выбирает язык программирования PascalABC.NET.(Хотя я с учениками на кружке изучаю Pyhton 3.4. А одиннадцатиклассник Марк Яковенко пишет игры на Pyhton 3.4)

Почему именно PascalABC.NET?

Языки программирования развиваются непрерывно. Здесь Паскалю не повезло. Заложенное еще его создателем, Никлаусом Виртом, стремление сделать язык простым, породило множество ограничений, сдерживающих развитие Паскаля. Язык оказался неспособным поддерживать современные идеи, методы и технологии программирования. Как следствие, в коммерческой сфере интерес к Паскалю со временем пропал, и язык остался востребованным лишь в сфере образования. В 2003 году энтузиастами Южного федерального университета (ЮФУ, г. Ростов-наДону) была создана учебная среда программирования Pascal ABC (ABC в английском языке имеет значение «азбука»), как альтернатива платным средам на базе языка Паскаль. Pascal ABC очень быстро набрал популярность в учебных заведениях России и стран бывшего СССР. Впоследствии разработчики решили дать Паскалю вторую жизнь. В процессе выработки спецификаций «нового Паскаля» была учтена потребность обеспечить быстрый последующий переход к изучению языка C#, а также шуточные пожелания преподавателей ЮФУ «писать в одну-две строчки» любые программы, которые даются в школах. В 2009 году появилась первая стабильная версия открытого проекта PascalABC.NET 1.2. За прошедшие годы произошли существенные изменения в коллективе разработчиков, язык также претерпел множество изменений. В данный момент актуальной является версия PascalABC.NET 3.4.

И всё-таки, почему же PascalABC ?

Базовый Паскаль за пределами сферы образования практически не используется. Можно возразить, что и PascalABC.NET также не используется за пределами сферы образования. Но если изучать Паскаль, то такой, на котором удобно и быстро писать и отлаживать программы.

1. PascalABC.NET позволяет в очень короткий срок перейти к изучению современных коммерческих языков программирования, например, C#, Pyhton 3.4. PascalABC.NET позволяет изучить современные технологии программирования, научиться оперировать последовательностями и кортежами, писать программы с элементами функционального стиля, применять стандартные коллекции.
2. PascalABC.NET дает возможность отказаться от концепции статических массивов в пользу динамических, существенно упрощая работу с массивами, в том числе, при обмене с процедурами и функциями.
3. Возможность решать в несколько строчек большинство задач, даваемых в учебниках по информатике, позволяет преподавателю сосредоточиться на алгоритмах решения, а не переписывать каждый раз одни и те же фрагменты реализации. Быстрое написание и отладка позволяют за одно занятие рассмотреть намного больше задач, повышая эффективность обучения.
4. Алгоритм, реализованный на PascalABC.NET нагляден и легко читается, делая лишним рисование блок-схем. Вносить изменения в готовую программу также быстро и легко. Дополнительным удобством является возможность использовать в программах идентификаторы, содержащие буквенные символы, отличные от латиницы (кириллические, греческие и т.д.).
5. Скорость выполнения готовой программы в большинстве случаев такая же, как у программы, написанной на C#.
6. В PascalABC.NET всегда остается возможность рассмотреть подробную реализацию нужного алгоритма на «низкоуровневом» базовом Паскале.

7. В PascalABC.NET можно обращаться к любым библиотекам платформы .NET Framework. Можно создавать свои библиотеки, которые затем использовать, в том числе, и в программах. написанных на других .NET-языках.

Подведём итог: PascalABC нужно изучать, хотя бы потому, что он до сих пор изучается в школах и университетах. Хотя бы потому, что на нём «завязаны» задачи из ГИА и ЕГЭ по информатике. Хотя бы потому, что Pascal может стать отличным подспорьем для начинающего программиста, который только знакомится с типами данных, переменными, циклами и другими явлениями из таинственного мира кодинга. Кроме того, здесь можно убить двух зайцев – познакомиться как процедурным (Pascal), так и с объектно-ориентированным (Delphi) программированием. Но я согласна и с теми, кто списал эти технологии на пенсию – это «мертвый» язык, которые используются исключительно в учебных целях. Но с этой ролью он справляется неплохо.

По большому счету, неважно, какой язык программирования выбрать для изучения ребенку 13-14 лет. Учить нужно не язык, а программирование. Мы должны развивать в школьниках логическое мышление, системный подход в решении задач – вот что по-настоящему важно. Мы должны всегда помнить, что язык – это средство, а не цель. И наша задача – выбрать правильное средство для достижения цели. Искусство программирования - это искусство мышления, не надо это забывать!

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

PascalABC.NET – современный мультипарадигменный язык программирования. Это означает, что на нем можно писать программы, сочетая различные парадигмы.

Под парадигмой понимается некий набор понятий, образующих стиль написания программы. Ранние универсальные алгоритмические языки базировались на императивной парадигме, предполагающей запись программы в виде набора инструкций, которые нужно последовательно выполнить для получения результата. Императивная парадигма отвечает на вопрос о том, как следует решать проблему.

Противоположностью является декларативная парадигма, в которой указывается, что задано и каким должен быть результат. Декларативная парадигма отвечает на вопрос о том, что нужно сделать для решения проблемы. Возможность сочетать в программе обе эти парадигмы делает программирование комфортным, позволяя не отвлекаться на мелочи. **Одна из первых предложенных парадигм – процедурная.** В процессе императивного программирования в последовательных участках кода выделяются некоторые блоки, к которым происходит обращение более одного раза. Эти блоки выделялись в языковые конструкции, получившие название процедур. В процедурной парадигме программа представляет собой набор процедур, одна из которых является главной и из нее производятся обращения к прочим.

Развитие процедурного программирования привело к возникновению парадигмы структурного программирования. В ее основе лежит представление программы, как совокупности процедурных блоков, которая имеет четкую иерархию. Это позволяет лучше видеть всю структуру связей между отдельными блоками, одновременно предполагая, что внесение изменений в один из них не влияет на работу остальных. **Парадигма объектно-ориентированного программирования** предполагает взгляд на программу, как на совокупность отдельных объектов, определенным образом взаимодействующих друг с другом. При этом, каждый объект создается на основе своеобразного «чертежа» - класса. **Все упомянутые парадигмы применяются в императивном программировании. Эт**им подходам противопоставляется парадигма функционального программирования, трактующая реализацию алгоритма, как процесса нахождения значений некоторых математических функций Возвращаясь к PascalABC.NET можно сказать, что у программиста есть возможность выбрать одну любую парадигму и придерживаться ее, либо в достаточно произвольной пропорции использовать в программе часть или даже все из вышеперечислен-

ных парадигм, получая при этом качественный и наглядный программный код с минимальными затратами труда.

Изучение языка программирования необходимо начать как можно раньше, по программе это происходит в 8 классе. В учебнике для 8 класса [Босова Л.Л., Босова А.Ю. Информатика: ФГОС.Учебник для 8 класса. – М.: БИНОМ. Лаборатория знаний, 2018] доступно раскрывается данная тема.

Немного теории для тех, кто самостоятельно изучает язык.

В языке Паскаль в простейшем случае программа начинается с английского слова begin (начало) и заканчивается словом end. (конец) – именно так, с точкой на конце. Конструкция begin … end называется операторными скобками, поэтому вполне допустимо сказать, что простейшая программа – это операторные скобки, за которыми следует точка.

```
Begin
 end.
```

Формат записи текста программы свободный, и это означает, что его размещение может быть произвольным, например, даже таким: begin end. Полезно знать, что у программистов сложился определенный стиль записи текстов программ, который облегчает восприятие программы и ее отладку.

Выделенные в тексте жирным шрифтом слова программы называются ключевыми (служебными словами, зарезервированными в языке для нужд программы). Конструкции языка Паскаль называются операторами.

Представьте простую задачу. Подсчитать сумму двух чисел.

Если числа небольшие, то можно устно или на калькуляторе, а если числа очень большие, то требуется подумать. Но ЭВМ «думает» намного быстрее и безошибочно. Программа может применяться для любых чисел.

Возьмем две независимые переменные

a, b – исходные данные

c – результат

Например: 10+8=18, -3+1000=997(целые)

9,3+8,1=17,4 (действительные числа)

Чтобы правильно работала программа, нужно выбрать тип переменных.

Типы переменных (числовых)

Основные типы данных:

- Целый integer - 32768 … 32767
- Длинный целый longint -2147483648 … 2147483647
- Вещественный real $2{,}9*10^{-39}$ … $1{,}7*10^{38}$

Дополнительные типы данных:

Форматы целого типа:

Название типа	Длина, байт	Диапазон значений
byte	1	0..255
shortInt	1	-128..+127
word	2	0..65535
integer	2	-32768..32767
longint	4	-2147483648..+2147483647

Форматы вещественного типа:

Название типа	Длина, байт	Количество значащих цифр	Диапазон десятичного порядка
real	6	11..12	-39..+38
double	8	15..16	-324..+308
extended	10	19..20	-4951..+4932
comp	8	19..20	-263+1..+263-1

Описание переменных происходит в разделе описания переменных:

```
var  переменная: тип;
Например: var x: integer;
var a, b, c: real;
var a, b: integer;
```

После определения типов переменных компилятор для каждой переменной отводит определенную ячейку памяти, где будут храниться некоторые данные. Например:

a	b	c
5	8	13

Чтобы компьютер производил вычисления, необходимо воспользоваться оператором присваивания.

Оператор присваивания

Имя переменной := выражение;

Например, c:=a+b

Сначала вычисляется выражение, затем полученной выражение присваивается переменной.

Чтобы компьютер знал над какими числами производить вычисления, необходимо ввести с клавиатуры значения переменных.

Оператор ввода

read ( ) – оператор ввода значений переменных (читать)

readln ( ) –оператор ввода значений переменных с переводом курсора на следующую строку (читать строку)

Например, **read (a)** или **read (a, b)**

read (a, b) – программа будет ждать, пока не введете значение переменных **a** и **b** через пробел

readln (a, b) – считывает значение **a** и **b,** затем переведет курсор на следующую строку.

Оператор вывода

write ( ) - оператор вывода значений переменных или фразы (писать)

writeln ( ) – оператор вывода значений переменных или фразы с переводом курсора на следующую строку (писать в строку)

Например, **write (c) –** выведет на экран значение переменной **c**

или **writeln (c)** - выведет на экран значение переменной и перейдет на следующую строку

write (a, b, c) – выведет значения переменных в строку без знаков препинания через позицию табуляции

write (‘мама’) – выведет слово, заключенное в апострофах

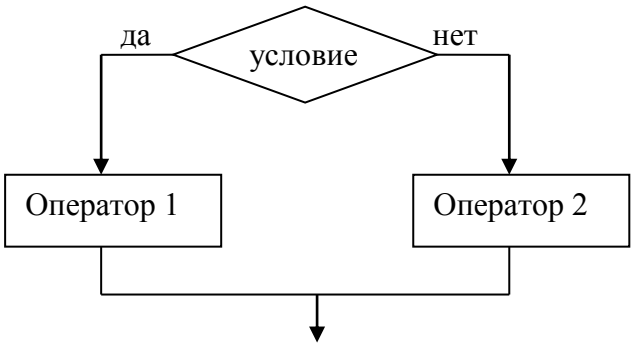
write (‘сумма равна ’, c) – выведет, например, **сумма равна 13**

writeln (a, b, c) – выведет значения переменных в строку без знаков препинания через позицию табуляции и переведет курсор на следующую строку.

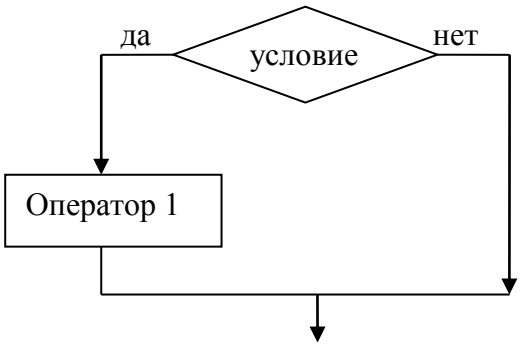
Запишем теперь полностью программу на Pascal.

```
Program summa;
var a, b, c: integer;
  begin
   writeln (‘введите слагаемые’);
   readln (a, b);
  c:=a+b;
writeln (‘сумма равна’, c);
 end.
```

Разветвляющимся называется алгоритм, позволяющий выбрать одно из нескольких возможных направлений решения задачи.



Полное ветвление



неполное ветвление

В PascalABC.NET ветвление реализуется с помощью ***условного оператора***.

полное ветвление:

if логическое условие **then** оператор 1
else оператор 2;

неполное ветвление:

if логическое условие **then** оператор 1;

Буквально перевести эту конструкцию можно так: "Если условие истинно, то выполнить оператор 1, иначе оператор 2"

Перед **else** точка с запятой не ставится!

Вначале проверяется логическое выражение, если оно верно, то выполняется Оператор 1, после этого выполняется часть программы, расположенная ниже, если условие ложно, то Оператор 1, не выполняется, выполняется оператор ниже.

Логическое условие может содержать одно или несколько условий с функциями and (и), or (или), not (не).

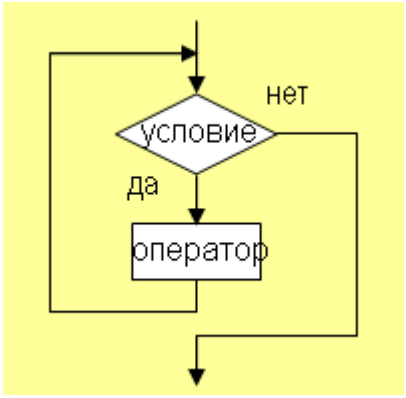
Например, условие для принадлежности числа x интервалу (0;10): **(x>0) and (x<10)**

Операторы 1 и 2 могут быть простыми и составными.

Если оператор составной (состоит из нескольких команд), то он заключается в логические скобки begin … end;

В PascalABC три вида циклов. ЦСикл сзаданным числом повторений (цикл со счетчиком), цикл с предусловием (**While**) и цикл с пост условием **(repeat).**

Второй вид циклов, используемых в Pascal - это цикл с предусловием.



Форма записи:

```
While условие        do оператор;
```

Пока условие истинно, выполняется оператор (или несколько), как только условие станет ложным, произойдет выход из цикла.

Условие проверяется в начале, перед циклом поэтому, если условие сразу же ложно, то цикл не повторяется ни разу.

После do ставить точку с запятой нельзя!

Если в теле цикла несколько операторов, то необходимо их заключить в операторные скобки **begin ... end** *(Если операторные скобки не поставить, то зациклена будет лишь первая операция)*

```
While условие do begin
                оператор 1;
                оператор 2;
                      ...
            end;
```

Переменные, участвующие в записи условия, должны изменяться в теле цикла, иначе может произойти зацикливание!

Задание 1. Какое значение примет переменная **x** в результате выполнения следующих фрагментов программ?

```
1) x:=1;
while x<10 do
x:=x+3;
x:=x+1;
```

В теле цикла только один оператор, так как нет скобок begin end. Построим трассировочную таблицу.

x	Условие
1	1<10 да
4	4<10 да
7	7<10 да

10	10<10 нет
10+1=11	

Ответ: x=11

2) x:=1;

```
while x<10 do begin
x:=x+3;
x:=x+1;
end;
```

В теле цикла только два оператора, так как есть скобки begin end. Построим трассировочную таблицу.

x	Условие
1	1<10 да
5	5<10 да
9	9<10 да
13	13<10 нет

Ответ: x=13

3) x:=1;

```
while x<>1 do begin
x:=x+3;
x:=x+1;
end;
```

Построим трассировочную таблицу.

x	Условие
1	1<>1 ложь

Ответ: x=1. Цикл не повторится ни разу, так как условие сразу ложно.

4) x:=50;

```
while x<100 do
begin
x:=x-10;
end;
```

Построим трассировочную таблицу.

x	Условие
50	50<100 да
40	40<100 да
30	30<100 да
	...

Ответ: Программа зациклится, так как условие никогда не станет ложно.

Задание 2. Составить программу, вычисляющую сумму натурального ряда чисел от 1 до 100

```
S = 1+2+3+4+…+100
```

```
i      1  2  3  4          100
1 способ: с помощью цикла for
s:=0;
for i:=1 to 100 do s:=s+i;
```

Параметр i увеличивается автоматически на 1 с каждым шагом.

2 способ: с помощью цикла while

Параметр i необходимо увеличивать на 1 в теле цикла. Также необходимо задать начальное значение параметра

```
i:=1;
S:=0;
while i<=100 do begin
                 S:=S+i;
                i:=i+1;
      end;
```

Сколько раз выполнится цикл? Для чего необходимо S:=0; i:=1? Что будет, если забудем begin end?

Задание 3. Составить программу для решения следующей задачи: Ввести целое число n с клавиатуры. Подсчитать сумму всех целых чисел от 1 до n с шагом 3.

S = 1+4+7+10+…+n

i = 1 4 7 10

```
read(n)
i:=1;
S:=0;
while i<=n do begin
          S:=S+i;
           i:=i+3;
    end;
writeln('S= ', S);
```

Задание 4. Составить программу для решения следующей задачи: Найти наибольший общий делитель NOD по алгоритму Евклида.

Например, NOD(20, 25)= NOD(20, 5)= NOD(15, 5)= NOD(10, 5)= NOD(5, 5)

```
while m<>n do if m>n  then m:=m-n
                  else n:=n-m;
writeln('NOD=',m)
```

Задание 8 ЕГЭ.(Часть 1)

Определите, что будет напечатано в результате работы следующего фрагмента программы

```
begin
       s:=0;
       k:=0;
      while k < 12 do begin
            s:=s+2*k;
            k:=k+3;
       end;
      write(s);
end.
```

Цикл while выполняется до тех пор, пока истинно условие k < 12, т. е. переменная k определяет, сколько раз выполнится цикл.

Так как числа небольшие, можно аккуратно выписать все s и k:

Построим трассировочную таблицу.

S	k	Условие
0	0	0<12 Да
0	3	3<12 Да
6	6	6<12 Да
18	9	9<12 Да
36	12	12<12 Нет

(Помните, что условие k < 12 проверяется сразу после k:=k+3, следовательно действие s:=s+2*k для k=12 выполняться не будет)

Следовательно, ответ — 36.

Второй вариант этого задания, если видна прогрессия, например:

Определите, что будет напечатано в результате работы следующего фрагмента программы:

```
var k, s: integer;
begin
       s:=0;
       k:=0;
      while k < 30 do begin
             k:=k+3;
            s:=s+k;
       end;
      write(s);
end.
```

Цикл while выполняется до тех пор, пока истинно условие k < 30, т. е. переменная k определяет, сколько раз выполнится цикл.

Так как последовательность k представляет собой арифметическую прогрессию, найдем n из неравенства:

$k_n = k_1 + (n-1)d < 30$, $k_1 = 0$, $d = 3$ (т. к. k:=k+3). Воспользовавшись методом интервалов, находим первое натуральное n, при котором нарушается условие: $n = 11$.

Значение s есть сумма первых n членов арифметической прогрессии. $b = \frac{2a_1 + (n-1)d}{2}n$, b — сумма первых n членов прогрессии, d — разность прогрессии, n — количество членов.

$$s = \frac{2s_1 + (n-1)d}{2}n = \frac{0 + (11-1)\cdot 3}{2}11 = 165.$$

Ответ: 165

Хотя все эти задачи можно решить простой трассировкой или хотя бы начать составлять таблицу и увидеть закономерность!

Определите, что будет напечатано в результате работы следующего фрагмента программы:

```
var n, s: integer;
begin
    n := 4;
    s := 0;
    while n <= 8 do
    begin
        s := s + n;
        n := n + 1;
    end;
    writeln(s);
end.
```

Поскольку изначально n = 4, цикл while n <= 8 выполнится пять раз. Следовательно, s = 4 + 5 + 6 + 7 + 8 = 30.

Ответ: 30.

Задание 20 ЕГЭ немного сложнее повышенный уровень, время – 5 мин
Тема та же: Анализ программы, содержащей циклы и ветвления.
Что ещёнужно знать:

- Алгоритм перевод чисел в другие системы счисления
- операции целочисленного деления (**div**) и взятия остатка (**mod**)
- как работают операторы присваивания, циклы и условные операторы в языке программирования
- базовые алгоритмы, алгоритм Евклида, сумма цифр числа и т.д.

Пример 20(1):
Получив на вход натуральное число x, этот алгоритм печатает два числа: a и b. Укажите наименьшее натуральное число, при вводе которого алгоритм печатает сначала 4, а потом –5.

```
var x, a, b: longint;
begin
readln(x);
a := 0; b := 1;
while x > 0 do begin
  if x mod 2 > 0 then
    a := a + x mod 9
  else
    b := b * (x mod 9);
  x := x div 9;
end;
writeln(a);
write(b);
end.
```

Решение :

1) Что делает предложенный алгоритм? Пока в числе *x*, переведенном в девятеричную систему счисления, есть хотя бы одна цифра, в цикле выполняются следующие действия: если число *x* нечетное (*x* mod 2 > 0), то переменная «a» увеличивается на последнюю цифру девятеричного представления числа x (*a* := *a* + *x* mod 9), иначе предыдущее значение «b» умножается на последнюю цифру девятеричного представления числа *x* (*b* := *b* * (*x* mod 9)). После этого отбрасывается последняя цифра девятеричного представления числа *x* (*x* := *x* div 9) и, если цифры в числе *x* ещё остались, то всё повторяется.
2) Для систем счисления с нечётным основанием (3, 5, 7, 9, ...) справедливо утверждение: число, записанное в системе счисления с нечетным основанием чётно тогда и только тогда, когда сумма всех его цифр чётна (поэтому судить о чётности числа по чётности его последней цифры в системе счисления с нечётным основанием нельзя).
3) Чтобы разобраться в том, что делает алгоритм, рассмотрим его работу при *X* $=8303_{10}$.
   а) Переведем 8303 в девятеричную систему счисления: $8303_{10} = 12345_9$.
   б) Находим сумму цифр: $X=12345_9$ -> *S* = 1+2+3+4+5=15; сумма нечётная -> число нечётное –> *a*=0+5=5;
   в) Следующий шаг: $X =1234_9$ -> *S* = 1+2+3+4=10. Сумма чётная –> число чётное –> *b*=1*4=4;
   г) $X =123_9$ -> *S* = 1+2+3=6. Сумма чётная –> число чётное –> *b*=1*4*3=12;
   д) $X =12_9$ -> *S* = 1+2=3. Сумма нечётная –> число нечётное –> *a*=0+5+2=7;

е) $X = 1_9$ -> Сумма нечётная –> число нечётное –> a=0+5+2+1=8;

ж) Таким образом, a=8, b=12. Будет напечатано сначала число 8, а потом – 12.

4) Решим нашу задачу. Наименьшее число должно начинаться с 1. Это даст нечётную сумму. Поэтому a=0+1=1. Чтобы получилось 4, нужна еще «3». Но 13 не подходит. Так как 1+3=4, то это будет четная сумма, и b станет равным b=1*4=4. Нам нужно b=5. Значит, 15 подойдет, так как 1+5=6 тоже четная сумма. Теперь можно добавить и «3». 153 даст нечетную сумму 1+5+3=9, поэтому a=0+1+3=4. Получилось число 153_9. Осталось перевести его в десятичную систему счисления: $153_9 = 129_{10}$.

Ответ: 129.

Задание 20 (2)

Ниже на четырёх языках записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: a и b. Укажите наименьшее из таких чисел x, при вводе которого алгоритм печатает сначала 3, а потом 2.

```
var x, a, b: integer;
    begin
    readln(x);
    a:=0; b:=0;
    while x>0 do
        begin
            a:=a + 1;
            if b < (x mod 8)
            then
                b:=x mod 8;
            x:=x div 8;
        end;
    writeln(a); write(b);
end.
```

Значение в переменной a после выполнения цикла равно количеству выполненных циклов. Поскольку требуется, чтобы программа напечатала сначала число 3, цикл должен выполниться три раза. Оператор div оставляет только целую часть от деления, следовательно, искомое число должно два раза делиться на 8 так, чтобы остаток был не меньше восьми. Следовательно, это число должно быть не меньше числа 64.

В переменную b записывается остаток от деления числа на 8. По условию требуется, чтобы после выполнения цикла переменная b имела значение 2, т. е. остаток от последнего деления на 8 в цикле должен быть равен 2.

Выполним программу для всех чисел, не меньших чем 64. Первое число, которое удовлетворит условию и будет наименьшим. Поскольку программа выводит целые числа и никаких других операторов к числу, кроме операторов div и mod, не применяется, будем рассматривать только целые числа.

При вводе числа 64 программа выведет числа 3 и 1. При вводе числа 65 программа выведет числа 3 и 1. При вводе числа 66 программа выведет числа 3 и 2. Следовательно, ответ 66.

Ответ: 66.

Получив на вход число x, этот алгоритм печатает число M. Известно, что x > 100. Укажите наименьшее такое (т.е. большее 100) число x, при вводе которого алгоритм печатает 26.

```
var x, L, M: longint;
begin
  readln(x);
  L := x;
  M := 65;
  if L mod 2 = 0 then  M := 52;
```

```
  while L <> M do     { * }
    if L > M then     { * }
      L := L - M      { * }
    else   { * }
      M := M - L;     { * }
  writeln(M);
end.
```

Решение:

1) видим, что в последней строке выводится на экран переменная **M**
2) ключевой момент решения: нужно узнать в строках программы, отмеченных знаком * в комментариях, АЛГОРИТМ ЕВКЛИДА для вычисления наибольшего общего делителя (НОД) чисел, записанный в переменные **M** и **L**
3) введённое значение **x** записывается в переменную **L** и участвует в поиске НОД
4) в переменную **M** до начала цикла записывается 65, но если было введено чётное (**L mod 2 = 0**) значение **x** (оно же **L**), значение **M** заменяется на 52
5) сначала предположим, что замены не было, и в M осталось значение 65; поскольку по условию алгоритм печатает 26, тогда получается, что **НОД(x,65)=26**; этого явно не может быть, потому что 65 не делится на 26
6) делаем вывод, что введено чётное значение **x** и произошла замена **M** на 52
7) итак, нужно найти чётное число **x**, большее 100, такое, что **НОД(x,52)=26**
8) первое число, большее 100, которое делится на 26 – это 104, но оно не подходит, потому что делится ещё и на 52, так что **НОД(x,52)=52**
9) поэтому берём следующее число, которое делится на 26: 104 + 26 = 130
10) Ответ: 130.

Второй аспект заключается в связи линии алгоритмизации и программирования с линией компьютера, с более глубоким раскрытием понятия программного управления ЭВМ. Ученики должны получить ответы на вопрос: что такое программа для ЭВМ? Как ЭВМ управляет «сама собой»? Почему ЭВМ можно назвать самоуправляемой системой?

При наличии небольшого объема учебного времени, программирование в базовом курсе может изучаться лишь на уровне введения. Основная задача ограничивается рамками все той же линии компьютера: раскрывается понятие программного управления работой компьютера. Изучение происходит на примерах простых программ. Показывается, как организуется простейший диалог компьютера с человеком: компьютер спрашивает, ученик отвечает, компьютер реагирует на ответ в соответствие с его содержанием. Показывается, как организуются простейшие вычисления, например, вводится числовая последовательность, выводится ее среднее арифметическое значение; или вводятся два числа, выводится их наибольший общий делитель (алгоритм Евклида) и т.п.

Изучение программирования - как прагматическая цель заключается в освоении основ профессионального программирования. Сегодня программирование на любительском уровне с практической точки зрения не представляет интереса. Используя прикладные программы можно сделать гораздо больше, чем с помощью языков программирования на ученическом уровне. Такую цель можно ставить только перед профильным или элективным курсом информатики.

Инструментальный характер программирования позволяет учащимся хорошо усвоить основные идеи алгоритмизации на практике, но этот подход требует много учебного времени. В классах гуманитарного и химико-биологического профиля на информатику выделяется только 1 час в неделю. В классах математического, экономиче-

ского и технологического профиля необходимо продолжение изучения технологий программирования. Как правило, у учащихся имеются базовые знания и на информатику в таких классах выделяется 2 и более часов в неделю. Здесь целесообразно построить обучение на сравнении различных информационных технологий решения задачи.

В преподавании информатики за счет школьного компонента усиливается раздел «Алгоритмизация и программирование». По моему мнению именно этот раздел целесообразно более подробно рассматривать в школе.

Раздел «Алгоритмизация и программирование» развивает алгоритмическое, операциональное мышление человека. Умение разбить задачу на подзадачи, умение воспользоваться готовым алгоритмом более простой задачи при решении сложной - это общеучебные умения и навыки, которые формируются у каждого выпускника на уроках информатики.

Таким образом, можно сказать, что раздел "Алгоритмизация и программирование" изучается на всех ступенях средней школы, но на разном уровне. В начальной школе происходит знакомство на интуитивном уровне с понятиями алгоритма, алгоритмических конструкций. В качестве учебных задач рассматривают бытовые, игровые, сказочные алгоритмы.

В средних классах школы в рамках данной темы происходит уточнение понятия алгоритма. При решении учебных задач учащиеся знакомятся с разными способами записи алгоритмов, изучают свойства алгоритма, рассматривают некоторые алгоритмы (алгоритм Евклида, сортировка данных и т.д.).

В старших классах, и особенно в классах физико-математического, информационно-технологического профилей, изучение этой темы строится в соответствии со Стандартом. Успешность учащихся в освоении этой темы во многом зависит от приобретенных ими общеучебных навыков в предыдущие годы обучения. Без сомнения, навыки, составляющие основу алгоритмического мышления, должны формироваться, начиная с младших классов.

Информатику в МОУ Пыщугская СОШ изучаем по УМК Босовой Л.Л. и УМК Семакина в 10 – 11 классах.УМК под редакцией Босовой Л.Л и Босова А. Ю. достаточно хорошо готовят к итоговой аттестации.

Задания дидактической линии «Алгоритмизация и программирование» охватывают следующую тематику:

- выполнение алгоритма для исполнителя с фиксированным набором команд;
- составление линейного алгоритма для формального исполнителя;
- вычисление значений переменных в линейных алгоритмах;
- циклические алгоритмы;
- обработка числовых массивов данных;
- алгоритмы обработки символьных и числовых последовательностей;
- составление программы на алгоритмическом языке или на языке программирования.

Тестовые задания данной дидактической линии предполагают знание учениками базовых алгоритмических структур, умение читать представленные с помощью них программы и владение навыками составления простейших алгоритмов . Задания можно условно разделить на задачи с использованием линейных алгоритмов, алгоритмов ветвления, циклических алгоритмов и их композиций на различных структурах данных.

Первые две темы требуют от ученика двух взаимообратных действий – выполнение заданного линейного алгоритма и собственно составление линейного алгоритма. При решении первого задания следует помнить, что при выполнении вычислений одна и та же переменная может, как находиться слева от знака присваивания, так и справа от него. Это означает, что в выражение подставляется «старое» значение переменной. При этом исходное значение переменной после вычислений заменяется «новым» полученным значением. При решении второго задания наоборот необходимо составить алго-

ритм для исполнителя по заданным для него линейным командам. Следует учитывать, что при решении данного тестового задания бывает удобным заменить заданные линейные команды противоположными инструкциями. Затем составить алгоритм, приводящий из конечной ситуации в начальную позицию. В ответ при этом записывается обратная последовательность команд для формального исполнителя.

Знание циклических алгоритмов предполагает вычисление суммы или количества значений для заданных переменных. Ученик должен внимательно выполнить, как правило, от шести от одиннадцати повторений цикла и записать в ответ значение переменной, выводимой на экран программы. При этом следует обращать внимание на исходные значения переменных, задаваемых перед циклом, на условие завершения цикла и на формульные зависимости в линейных командах, изменяющих значения заданных величин.

Задание на обработку чисел в одномерном массиве подразумевает понимание школьника, что эта структура объединяет множественные однородные данные и обеспечивает более удобный способ доступа к ним. Для этого не надо объявлять множество различных переменных, следует задать массив и обрабатывать значения его элементов, указывая их позиции в нем. При выполнении такого задания среди указанного набора данных, как правило, необходимо найти позицию либо значение первого или последнего минимального или максимального элемента массива, найти количество либо вычислить сумму элементов массива, значения которых меньше или больше заданных. Другими словами осуществить по заданному условию среди элементов массива отбор и выполнить с этими элементами операции, представленные в тестовом задании. Таким образом, это тестовое задание характеризует умение школьника осуществлять выбор среди множества заданных числовых значений, а отличительной его чертой выступает умение переформулировать при его решении алгоритмические записи на языке программирования в содержательный контекст задачи.

ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ, КОТОРЫЕ ИСПОЛЬЗУЮТСЯ В ПРОЦЕССЕ ПОДГОТОВКИ К ЕГЭ.

Для подготовки к ЕГЭ мной используются образовательные технологии: и обучение в сотрудничестве, и проблемное обучение, игровые технологии, технологии уровневой дифференциации, групповые технологии, технологии развивающего обучения, технология модульного обучения, технология проектного обучения, технология развития критического мышления учащихся, ИКТ технологии и др.

Технология критического мышления

Применение методики формирования критического мышления приводит к изменению структуры урока. **Выделяются три основные стадии:**

1. вызов,
2. осмысление,
3. размышление (рефлексия).

Основная задача стадии вызова – пробудить интерес, подготовить учащихся к предстоящей работе. На этой стадии озвучивается цель урока, учащиеся ее принимают, происходит мотивация их дальнейшей деятельности. На стадии осмысления учащиеся сталкиваются с новой информацией; они пытаются решить поставленную проблему, опираясь на сведения, предоставленные учителем, текст учебника или документа. На стадии рефлексии происходит корректировка взглядов учащихся на основании полученной ими новой информации, присвоение нового знания. Школьники высказывают собственные идеи и аргументируют их. Правила проведения уроков по формированию критического мышления

• В работу должны быть вовлечены все учащиеся. Для этого, например, используют методический прием – короткие выступления при обсуждении темы.

• Следует позаботиться о психологической подготовке учащихся. Для этого полезно проводить разминки, поощрять учеников за активное участие в работе, предоставлять им возможность самореализации.
• Учащихся делятся на группы по 5–6 человек. Только при этом условии возможна продуктивная работа в группах. Очень важно, чтобы каждый был услышан, каждая группа имела возможность выступить по проблеме.
• Процедуру и регламент урока надо обсудить в начале занятия и не нарушать их.
• Ученики могут делиться на группы добровольно, но обязательно надо добиться, чтобы группы были примерно равны по силам.
Методы и приемы, работающие на эту технологию через уроки информатики. Мозговой штурм
При работе нужно обращать внимание на иерархию вопросов, которые сопровождают каждый этап «Мозгового штурма»:
•I уровень что ты знаешь?
•II уровень как ты это понимаешь? (применение других знаний, анализ)
•III уровень применение, анализ, синтез

Технология интерактивного обучения.

Данная технология основана на модели обучения, включающей три основных этапа:
1. осмысление нового материала: представление информации из различных источников, организация работы с информацией, поддержка обратной связи;
2. интерактивное задание предполагающее цепочку действий: индивидуальное продумывание – групповое обсуждение и графическое представление. Интерактивные задания позволяют использовать различные источники информации, в том числе собранные во время группового и фронтального обсуждения для самостоятельного выполнения задания;
3. обобщение, апробирование и рефлексия: использование новых представлений для анализа практической ситуации, обоснование способа решения проблемы, систематизация представлений.

ИКТ-технология.

Данная технология применяется как при решении заданий 2 части при организации самостоятельной работы с Интернет – сервисами и телекоммуникационной системы СтатГрад, работу в облаке знаний, на сайте Д. Гущина «Решу ОГЭ» и др.

С заданиями 24, 25 ЕГЭ выпускники справляются хорошо, а задание 27 без дополнительной подготовки решить невозможно - на это требуется очень много времени.

Часть 2 КИМов содержит 4 задания, первое (№24) из которых повышенного уровня сложности, остальные 3 задания высокого уровня сложности. Задания этой части подразумевают запись развернутого ответа в произвольной форме.

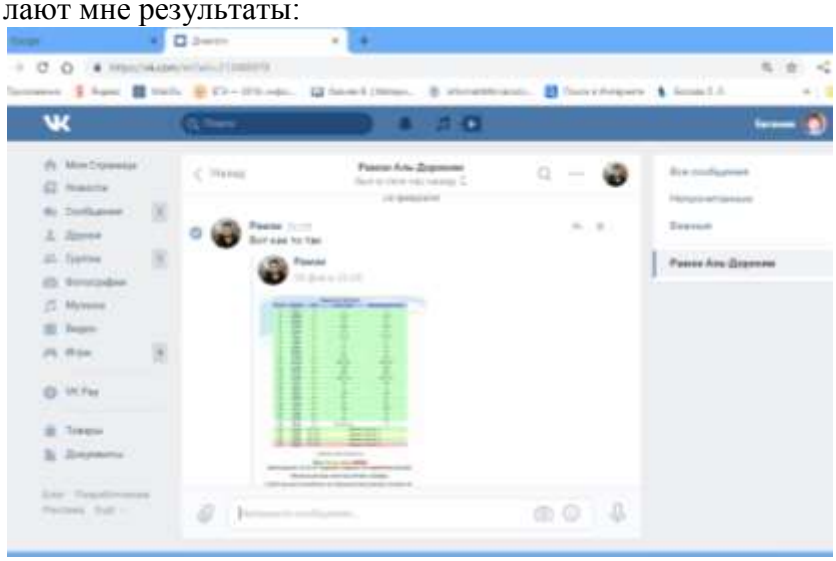
Задания части 2 направлены на проверку сформированности важнейших умений записи и анализа алгоритмов. Эти умения проверяются на повышенном и высоком уровнях сложности. Также на высоком уровне сложности проверяются умения по теме «Технология программирования».

В КИМ ЕГЭ элементы теории алгоритмов составляют 17%, а элементы программирования– 25% всей работы.

Вторую часть заданий на программирование мы с ребятами разбираем отдельно, после того, как успешно был выполнен итоговый тест по первой части. И в этом нам очень хорошо помогает «Раздаточный материал К. Ю. Полякова – материалы для подготовки к ЕГЭ», пособие «Отличник. Информатика», сайт «Решу ЕГЭ» Дмитрия Гущина и беседы в Контакте.

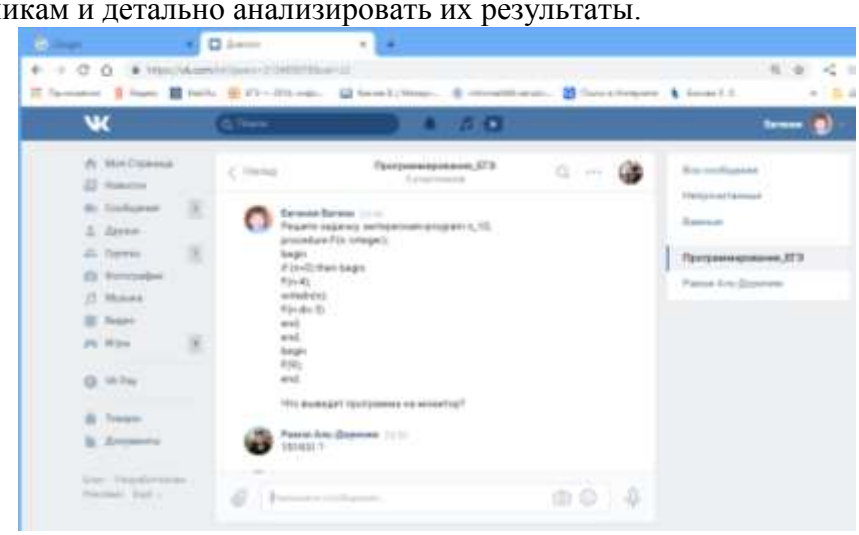
Решение сложных задач», где очень подробно разобраны задания 2 части ЕГЭ и представлены задачи для тренировки, а также собственный банк задач по 2 части.Так как урока не хватает для сложных задач, в группах Vk.com создала на своей страничке беседу «Подготовка к ЕГЭ», в этот год собираются сдавать ЕГЭ 5 человек. Их пригла-

сила в группу и там разбираем все задания повышенного уровня сложности, присылаю им задания , которые составляю с помощью сервиса examer.ru. А они решают, и присылают мне результаты:



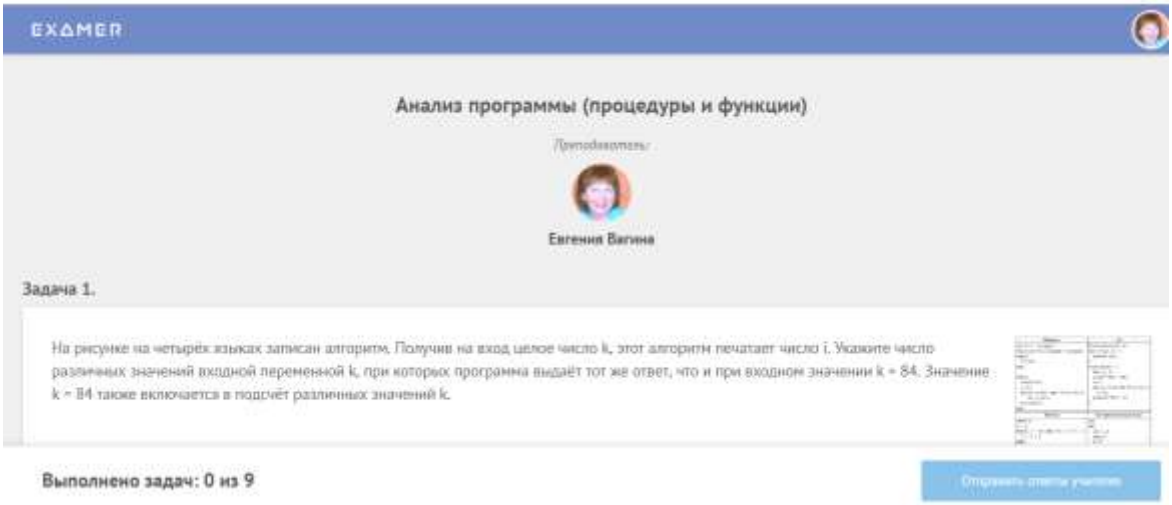
Важную роль в методике обучения программированию, следует отводить самостоятельной работе учеников, так как только самостоятельная разработка алгоритмов и программ, должным образом способствует развитию алгоритмического мышлению и закреплению необходимых навыков.

В Интернете есть сервисы, помогающие учителю, мы используем их для подготовки к ЕГЭ. Это EXAMER.RU(https://examer.ru/app/intro) сервис, который поможет сэкономить время на составление тестов для ваших учеников и проверку их результатов. С его помощью вы можете удобно создавать индивидуальные тесты, выдавать их ученикам и детально анализировать их результаты.



https://teacher.examer.ru/app/test/23a1a

Анализ программ Процедуры и функции Подготовка к ЕГЭ на examer.ru



После разбора заданий 2 части составляю итоговую работу, где представлены различные задания из частей 1 и 2.

МЕТОДИЧЕСКАЯ ЧАСТЬ

Когда речь идет о технологии подготовки к экзамену, прежде всего нужно отметить, что нет единого универсального решения, а есть типовые варианты, из которых учитель может подобрать себе подходящий. Учитель выступает организатором процесса, обеспечивая его системность, содержательную часть, консультационную и контролирующую.

Следует равномерно распределить силы учащегося и, скорее всего, создать возможность для дополнительных занятий, то есть разработать план подготовки к сдаче ЕГЭ по информатике и ИКТ с учетом индивидуальных особенностей учащегося или группы учащихся. Какое бы мнение педагоги не имели о ЕГЭ, приходится работать в рамках существующих обстоятельств и принимать решения: как готовиться к экзамену продуктивно, как создать условия для успешной сдачи экзамена выпускниками и самое главное самим быть готовыми к ЕГЭ содержательно, методически и организационно. Я думаю, что для предметов технического уровня форма экзамена в виде ЕГЭ вполне приемлема.

У меня сложилась определенная система подготовки учащихся к итоговой аттестации. Большое внимание в своей работе уделяю самообразованию для грамотной и квалифицированной подготовки учащихся к ЕГЭ. Только системная работа в течение учебного года позволяют повысить продуктивность и качество подготовки к ЕГЭ и даст шанс надеяться на положительные результаты сдачи экзамена.

Опрос желающих сдавать ЕГЭ нужно провести как можно раньше. Не секрет, что времени на сдачу экзамена порой не хватает. (ЕГЭ по информатике и информационно-коммуникационным технологиям (ИКТ) продолжается 3 часа 55 минут (минут)). Это объясняется разными причинами: природными качествами отдельных учащихся, например, медлительностью, качеством знаний и умений учащихся по информатике и математике, уровнем сложности задач ЕГЭ. Безусловно, время учениками было потеряно на то, чтобы справиться с волнением. Поэтому тестирование учащихся провожу почти на каждом уроке с ограничением времени. В сентябре в 11 классе провожу диагностический тест за курс 9 – 10 классов, который позволяет выявить проблемы в разных областях. На основании чего мною разрабатываются программы дополнительных занятий.

Только системная работа в течение учебного года позволяет повысить продуктивность и качество подготовки к ЕГЭ.

Работу по подготовке к экзамену в формате ЕГЭ можно разбить на несколько частей.

Первая состоит в анализе результатов предыдущего ЕГЭ.

Вторая состоит в том, что начиная с 10-го класса, в планы уроков вносятся изменения, ориентированные на подготовку к ЕГЭ практически на каждом уроке (5 – 6 задач из ЕГЭ по изучаемой теме).

Третья часть предполагает разработку программы дополнительных занятий, по подготовке выпускников непосредственно к сдаче экзамена.

Планы уроков, начиная с 8-го класса, должны заканчиваться пунктом "Примеры заданий из ЕГЭ". Желательно при закреплении материала на уроке давать контрольные вопросы и задания в стандартном формате, соответствующем ЕГЭ.

После прохождения какой-то темы, которая объединяет в себе несколько уроков, я провожу контроль знаний. Контроль состоит из заданий, подобных заданиям ЕГЭ. Тестирование можно проводить в бумажном или электронном виде, тексты тестов и задания составляю, используя многочисленную литературу с готовыми текстами тестов по

основным разделам базового курса. Стараюсь выбирать задания из имеющихся на сегодняшний день в базе данных контрольно-измерительных материалов (КИМ) для проведения ЕГЭ по информатике, из всевозможных демонстрационных, репетиционных и реальных вариантов ЕГЭ, а также из сборников для подготовки к ЕГЭ, допущенных Министерством образования и науки. Моя задача при подготовке к урокам — выбрать из имеющегося материала задания, соответствующие теме урока.

В начале учебного года составляю план работы по подготовке выпускников 11 классов к ЕГЭ.(план представлен в виде таблицы):

Муниципальное общеобразовательное учреждение Пыщугская средняя общеобразовательная школа
Вагина Евгения Николаевна учитель информатики и ИКТ

План
работы учителя информатики
по подготовке учащихся 11 классов к ЕГЭ по информатике в 2016 – 2017 учебном году

Направление	Мероприятие	Сроки выполнения
1.Информационное обеспечение деятельности учителя информатики	**I. Информационная деятельность**	
	1. Информировать обучающихся и родителей 11 классов об особенностях государственной (итоговой) аттестации в 2016 – 2017 учебном году	По мере поступления материалов
	2. Информировать обучающихся 11 классов об адресах сайтов в Интернете, где размещены материалы по подготовке и проведению ЕГЭ по поступлению в ВУЗ.	
	3. Оформить уголок в кабинете для подготовки к ЕГЭ по информатике (дидактический материал, демоверсии, образцы решений заданий разного типа и т.д.)	В течение года
	4. Создать банк материалов по подготовке к ЕГЭ	
	5. Информировать родителей обучающихся 11 классов о промежуточных результатах подготовки к ЕГЭ	
	6. Составить рекомендации для учащихся по подготовке к ЕГЭ по информатике	
	7. Информировать родителей о сборниках по подготовке к ЕГЭ, сайтах Интернета с КИМами и тестовыми тематическими заданиями, ВУЗов.	Через родительские собрания
2. Анализ, диагностика, мониторинг освоения учащимися классов предмета «Информатика»	**II. Аналитико - диагностическая деятельность**	
	1. Провести анализ успеваемости учащихся по информатике за 10 класс	Сентябрь
	2. Провести вводную диагностическую работу по материалам ЕГЭ для определения проблем учащихся в освоении тем	Октябрь
	3. Систематизировать затруднения и пробелы в знаниях учащихся по информатике	Октябрь

	4. Проводить анализ успеваемости учащихся 10 классов по информатике в течение учебного года.	В течение года
	5. Вести диагностические карты подготовки к итоговой аттестации учащихся 11 классов по информатике	
	6. Вести мониторинг и анализировать результаты самостоятельных, проверочных, плановых диагностических работ по информатике учащихся 11 классов	
	7. Провести репетиционный экзамен по информатике ЕГЭ	Март 2017г.
	8. Провести анализ результатов ЕГЭ по информатике учащихся 11 классов	Июнь 2017г.
3.Организация и проведение дополнительных занятий и консультаций	**III. Учебная и консультационная деятельность**	
	1.Проводить дополнительные занятия для учащихся, мотивированных на получение хорошего результата на ЕГЭ по информатике	1 раз в неделю (четверг)
	2. Проводить дополнительные занятия со слабоуспевающими обучающимися	
	3.На уроках информатике дополнительно уделить время изучению тем, отображаемых в ЕГЭ	Во время проведения уроков информатики
	4. На уроках информатике включить разделы для повторения ранее изученных тем	
	5. Проводить индивидуальные консультации для учащихся классов	По мере необходимости
	6. Тренировать учащихся 11 классов работать с бланками ЕГЭ	В течение года

Подготовка к ЕГЭ 2017 – 2018 учебный год

№ п/п	Ф.И.	Результаты входной диагностической контрольной работы	Контроль тест №1	Контроль тест №2	Контроль тест №3	Контроль тест №4	Контроль тест №5	Контроль тест №6	Контроль тест №7	Результаты репетиционного экзамена	Результаты экзамена	
											Оценка	Балл
1.												
2.												
3.												
4.												
5.												
6.												

Обязательно провести мониторинг входной диагностической контрольной работы

Входная диагностическая контрольная работа (см. приложение 1)

11 классы

№ п/п	Ф.И.	задания											Итого	Оценка
11														
1														
2														
3														
4														
5														

С целью контроля прохождения всех заданий, а также наглядной картины «готовности» ученика к ЕГЭ следует проводить мониторинг каждого сдающего экзамен ученика. Таким образом, можно получить достоверную картину успехов каждого ученика, а ученик, свою очередь, узнает уровень своей подготовленности. Все результаты заношу в «Журнал по подготовке к ЕГЭ», чтобы вести рефлексию

II.Работа с родителями по обеспечению КИМ по информатике, ознакомлению родителей с текущей успеваемостью и результатами тестов, посещаемостью индивидуальных занятий

В конце учебного года в 11 классе для подготовки учащихся к государственной итоговой аттестации организую повторение курса программирования по плану:

Повторение за месяц перед экзаменом

№ занятия	Тема занятия	Рассматриваемые вопросы
1.	Решение задачи 8 ЕГЭ	Повторяем основные конструкции языка программирования, понятия переменной, оператора присваивания, операторы div, mod, виды ветвлений, циклы, часо применяемые алгоритмы: выделение цифр числа, перевод в двоичную, троичную и т.д. системы, алгоритм Евклида, вычисление факториала, последовательности (Фибоначчи и др.)
2.	Решение задачи 11 ЕГЭ	Повторяем функции и процедуры, рекурсивные алгоритмы
3.	Решение задачи 19 ЕГЭ	Виды массивов и их обработка (заполнение, считывание, поиск, сортировка, и др.)
4.	Решение задачи 20 ЕГЭ	Анализ программ с циклами и условными операторами Анализ алгоритма, содержащего вспомогательные алгоритмы, цикл и ветвление
5.	Решение задачи 21 ЕГЭ	Анализ программ, использующих процедуры и функции Анализ программ с циклами и подпрограммами
6.	Решение задачи 22 ЕГЭ	Умение анализировать результат исполнения алгоритма
7.	Решение задачи 24 ЕГЭ	Запись программы на языке программирования и исправление допущенных ошибок
8.	Решение задачи 25 ЕГЭ	Написание короткой (10-15 строк) программы (например, обработки массива) на языке программирования
9.	Решение задачи 27 ЕГЭ	Создание собственных программ. Записи, обработка файлов, строки и символы (30-50 строк) для решения задач средней сложности

Вывешиваю на стенде и знакомлю учащихся с основами психологической подготовки к экзамену.

Подготовка к экзамену:

Подготовьте место для занятий: убери со стола лишние вещи, удобно расположи нужные учебники, пособия, тетради, бумагу, карандаши и т.п.

- Составьте план занятий. Для начала определите: кто вы - «сова» или «жаворонок», и в зависимости от этого максимально используйте утренние или вечерние часы. Составляя план на каждый день подготовки, необходимо четко определить, что именно сегодня будет изучаться. Не вообще: «немного позанимаюсь», а какие именно разделы и темы.
- Чередуйте занятия и отдых: 40 минут занятий, затем 10 минут - перерыв. Во время перерыва можно помыть посуду, полить цветы, сделать зарядку, принять душ.
- Выполняйте как можно больше различных опубликованных тестов по этому предмету. Эти тренировки ознакомят Вас с конструкциями тестовых заданий.

- Тренируйтесь с секундомером в руках, отмечайте время выполнения тестов
- Готовясь к экзаменам, мысленно рисуйте себе картину триумфа. Никогда не думайте о том, что не справитесь с заданием.
- Оставьте один день перед экзаменом на то, чтобы еще раз повторить самые трудные вопросы.

Накануне экзамена

Многие считают: для того, чтобы полностью подготовиться к экзамену, хватает всего одной, последней перед ним ночи. Это неправильно. Устали, и не надо себя переутомлять. Напротив, с вечера совершите прогулку, перед сном примите душ. Выспитесь как можно лучше, чтобы встать с ощущением «боевого» настроя.

Одним из направлений организационно-методической работы является создание банка тестовых заданий, подбор учебно-методической литературы. Учащимся нравится такой метод контроля знаний как тестирование. Его можно проводить в бумажном или электронном виде, тексты тестов и задания составляю, используя многочисленную литературу с готовыми текстами тестов по основным разделам базового курса.

Широкое использование систем тестового контроля не только позволяет подготовить учащихся к формату письменных экзаменов, проводимых в виде тестов, но является несомненным подспорьем на уроках информатики. Такие тесты могут носить не только контролирующие, но обучающие и закрепляющие функции, служить для осуществления как текущего или промежуточного, так и тематического или итогового контроля знаний.

Для того чтобы добиться хороших результатов при прохождении государственной (итоговой) аттестации в будущем, провожу итоговое тестирование по темам, разделам программы по информатике, составляя их при помощи тестовой оболочки «MyTest», составляя тесты сама и используя готовые тесты.(http://mytest.klyaksa.net/). См. приложение 1.

MyTest - это система программ (программа тестирования учащихся, редактор тестов и журнал результатов) для создания и проведения компьютерного тестирования, сбора и анализа результатов, выставления оценки по указанной в тесте шкале.

Программа легка и удобна в использовании. Программа MyTest работает с девятью типами заданий: одиночный выбор, множественный выбор, установление порядка следования, установление соответствия, указание истинности или ложности утверждений, ручной ввод числа, ручной ввод текста, выбор места на изображении, перестановка букв. В тесте можно использовать любое количество любых типов, можно только один, можно и все сразу. В заданиях с выбором ответа (одиночный, множественный выбор, указание порядка, указание истинности) можно использовать до 10 (включительно) вариантов ответа.

Программа MyTest X распространяется бесплатно.

На протяжении всего периода обучения наряду с контрольными работами: теоретическими и практическими, практикую тесты в различных видах и формах: на бумажном носителе или компьютерное тестирование (2-3 и более вариантов). Это даёт положительные результаты. За последние три года увеличилось количество детей, принимающих участие в интернет – проектах, конкурсах, олимпиадах по информатике Стараюсь выбирать задания из всевозможных демонстрационных, репетиционных и реальных вариантов ЕГЭ, а также из сборников для подготовки к ЕГЭ, рекомендованных ФИПИ.

Если ученик выполняет тест меньше 50%, то он устно к следующему занятию готовит теоретическую часть и готовится к практической части - тесту. Если же ученик и во второй раз показывает такой же результат, то в индивидуальном порядке происходит разбор тех тестовых заданий, в которых допущены ошибки.

Главной задачей обучения информатике считаю достижение оптимального уровня развития мыслительных способностей каждого ученика. При подготовке к урокам я подбираю задания так, чтобы они вызывали эмоциональный отклик у детей, пробуждали интерес. Хорошим считаю такой урок информатики, на котором ученик познаёт себя, делает открытия, ищет верные решения, сомневается, радуется своим успехам и успехам своих товарищей. При изучении нового материала использую метод поисковых или частично-поисковых ситуаций.

Одной из проблем при изучении курса информатики является разный уровень знаний ребят. Для кого-то не составит труда выполнить задание (справляются очень быстро), а кто-то осваивают его очень медленно. В таких случаях стараюсь составить здания разных уровней сложности. Для этого систематизирую наработанные материалы разных лет по разделам экзаменационной работы, используя УМК различных авторских коллективов, материалы сети Интернет и различных электронных пособий и CD для подготовки к ЕГЭ.

С целью подготовки к ЕГЭ рекомендую учащимся участвовать во всероссийских олимпиадах по программированию, во всероссийском конкурсе «КИТ», «Инфознайка», олимпиаде УРФО по основам наук, других конкурсах. Это позволяет им проверить качество своих знаний по информатике, оценить скорость решения ими задач, отработать внимательность при заполнении бланка с ответами.

Сдать экзамен по информатике успешно под силу только учащимся с хорошим логическим мышлением. И моя цель максимально развить это логическое мышление, повысить интерес к предмету.

В ходе подготовки должна ставиться задача не только успешной сдачи учащимися экзамена, но и набор ими при этом максимально возможного количества баллов: каждым из учеников в зависимости от уровня его подготовки.

Опираясь на свой многолетний опыт подготовки выпускников к экзаменам и на их результаты, я могу предложить некоторые рекомендации педагогам по подготовке учащихся к ЕГЭ по информатике:

- подготовку к экзамену в формате единых государственных экзаменов необходимо начинать с 7 класса, включая в тесты для проверки знаний задания из ЕГЭ. А изучением основ алгоритмизации и программирования можно заниматься во внеурочное время;
- в классах, изучающих информатику на базовом уровне, проводить тренинговые элективные курсы, например, «Готовимся к ЕГЭ по информатике»;
- ни в коем случае не натаскивать детей на определенные типы тестовых заданий из ЕГЭ. Это приведет к тому, что увидев «незнакомую» задачу, ребенок растеряется и даже не попытается ее решить. Целесообразнее, давать основы объемных тем, таких как «Системы счисления», «Кодирование информации», «Логика», «Алгоритмизация и программирование», и проводить контроль знаний, каждый раз включая новый тип задач. При этом обязательно вести рефлексию: если ученики выполняют меньше 80 % заданий, необходимо продолжить изучение темы;
- создать собственную рабочую коллекцию полезных ссылок на основные Интернет-источники с материалами для пополнения своей методической и дидактической копилки, а также набор методических пособий, рекомендованных ФИПИ для подготовки к экзамену;
- постоянно пополнять систематизированный материал разных лет по разделам экзаменационной работы новыми типами задач и возможными способами их решения.
- И самое важное, каждому учителю, занимающемуся подготовкой учащихся к единым государственным экзаменам, необходимо помнить, что только системная рабо-

та в течение нескольких лет позволит повысить продуктивность и качество подготовки к ЕГЭ и даст шанс надеяться на положительные результаты сдачи экзаменов.

Система развивающих заданий, призванная вызвать интерес учащихся и, как следствие, успешной сдачи ЕГЭ, с моей точки зрения возможно при использовании следующих методов:

1) развитие самостоятельности в освоении предмета,

2) введение творческих заданий,

3) использование дифференцированных заданий,

4) использование игровых методик для закрепления, повторения и проверки изучаемого материала.

Подготовка к итоговой аттестации учащихся в новой форме – это длительный и кропотливый, в какой-то степени творческий труд, требующий помощи и консультации со стороны педагога и столь же вдумчивой и напряженной работы ученика.

Подготовку к экзамену можно проводить как в рамках урока, так и во внеурочное время (через элективные курсы, факультативы, индивидуальные занятия и консультации), а также через дистанционное обучение. Дистанционные образовательные технологии (ДОТ) - это образовательные технологии, реализуемые в основном с применением информационных и телекоммуникационных технологий при опосредованном (на расстоянии) или не полностью опосредованном взаимодействии обучающегося и педагогического работника (дистанционного учителя, преподавателя).

Использование ИКТ при подготовке к ЕГЭ имеет следующие преимущества:

- организация самостоятельной работы учащихся;
- индивидуализация обучения;
- рост объёма выполненных заданий;
- повышение мотивации и познавательной активности за счёт разнообразия форм работы;
- поддерживание учителей в состоянии творческого поиска новых методов обучения;
- проявление интереса учащихся к предмету;
- создание собственного банка учебных и методических материалов, готовых к использованию в учебно-познавательном процессе.

Наилучший способ достижения хорошего результата – это ежедневная и разнообразная тренировка не только различных заданий, но и с ограничением во времени. Компьютер может использоваться на всех этапах обучения: при объяснении нового материала, закреплении, повторении, контроле.

При подготовке к ЕГЭ мы применяем электронные пособия, презентации, тестовые работы, ресурсы Интернета, видео-уроки, Skype.

Большую популярность при подготовке к экзамену получили электронные пособия. Учащимися совместно с учителем ведётся разработка серий электронных пособий для подготовки к ЕГЭ по соответствующим разделам. Пособия содержит теоретический материал, далее идут задания без выбора ответа, данные задания решаются вместе с учителем. Далее идут тесты, разделённые по уровню трудности, с выбором ответа. Очень большим плюсом является то, что учитель и учащийся сразу же после прохождения теста, видят результат. Если учащийся выбирает неправильный ответ, то он получает комментарий с названием тем, которые необходимо ему повторить. Пройти на следующий уровень тестирования можно только тогда, когда предыдущий тест выполнен на 100%. Пособия содержат домашнее задание и видео с объяснением учителем

теоретического материала и с решением некоторых заданий. Таким образом, учащиеся, используя электронные пособия, могут самостоятельно изучать теоретический материал и осуществлять контроль над усвоением материала и в домашних условиях.

Кроме электронных пособий используются презентации. Презентации очень хорошо реализуют принцип наглядности, сокращает время обучения, можно неоднократно возвращаться к пройденному материалу. При прохождении нового материала по некоторым темам свой вклад в создание презентации вносят и сам учащиеся. Ученики заранее готовят материал, ведут поиск в сети Интернет и по другим источникам. На уроке они выступают с объяснением материала, используя свои презентации. Самостоятельно готовя презентации, учащиеся приобретают следующие навыки:

- находить нужную информацию и систематизировать её;
- выделять главное, устанавливать связи между частями;
- находить ошибки в получаемой информации;
- попробовать себя в роли организатора.

При подготовке к ЕГЭ огромную роль играет и использование Интернета. Интернет прочно вошёл в нашу жизнь. К нему мы обращаемся в поисках дополнительного материала к уроку. Мы используем on-line тесты при подготовке к экзамену. И уже не надо тратить много времени на проверку тестов. За короткое время мы получаем объективную картину уровня усвоения изучаемого материала и имеем возможность вовремя скорректировать. Есть возможность выбора уровня трудности задания для конкретного ученика. Очень важно то, что ученик, после выполнения теста сразу видит результат с указанием ошибок.

Практикуется и использование тематических тестов. Тесты предназначены не только для проверки усвоения пройденного материала. Но и для изучения отдельных тем. Многие задания снабжены решениями, которые можно просмотреть и во время выполнения работы, так и после прохождения теста.

При подготовке к ЕГЭ я используем следующие интернет-ресурсы: http://fipi.ru/, http://www.ege.edu.ru/, http://www.ege.ru/, http://kpolyakov.narod.ru/school/ege.htm, https://inf-oge.sdamgia.ru/tes t http://ege.yandex.ru/informatics, http://college.ru/informatika/, https://www.imumk.ru/player/1610

Использование их открыло новые возможности при подготовке к экзамену. Работа происходит в режиме общения между учителем и учащимся. Учитель прикрепляет задания в беседе ВК. Если у ученика есть вопрос, то он может задать его своему учителю и получить ответ.

Я уверена, что использование роль ИКТ при подготовке к ЕГЭ будет расти с каждым годом и данный рост будет вполне оправданным. Ещё Ушинский отмечал, что знания будут тем прочнее и полнее, чем большим количеством органов чувств они воспринимаются. Все эти условия реализуются при использовании ИКТ при подготовке к ЕГЭ. Таким образом, использование ИКТ позволяет повысить уровень знаний, облегчает подготовку к ЕГЭ, делает уроки нетрадиционными, запоминающимися, интересными, более динамичными. Но несмотря на большие плюсы, нельзя забывать, что компьютер – это всего лишь средство, которое способствует достижению поставленных целей и задач урока, но компьютер никогда полностью не заменит учителя.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО РЕШЕНИЮ ЗАДАЧ

Рассмотрим более подробно некоторые типичные ошибки учащихся, выявленные при подготовке к ЕГЭ по темам программирования.

Дети приходят к нам на урок, чтобы научиться программировать. Все они хотят стать классными и востребованными специалистами, зарабатывать кучу денег и заниматься любимым делом. И это – никакой иронии! – очень похвально. Действительно, профессия программист – это, пожалуй, лучшая карьерная перспектива на сегодняшний день.

И все начинающие программисты, наши ученики да и мы с вами, дорогие учителя, совершают одинаковые ошибки. Самые частые из них – перед вами. Хочешь стать программистом? Начинай свой путь без ошибок!

Ошибка №1. Неправильный выбор языка или технологии.

Здесь сразу нужно сделать замечание – как правило, это не является ошибкой в строгом смысле слова. «Вина» за неправильный выбор курса или языка программирования чаще всего лежит не на самом ученике, а на внешних обстоятельствах – отсутствии знаний и опыта, мимолетном увлечении, переоценке своих возможностей. Поясню: иногда у нас учатся ребята, которым нравится звучание слова Java или Pyhton, но не имеющие ни малейшего представления о программировании. Они слышали, что на сегодня это одна из самых востребованных технологий (и это правда, учитывая взрывную популярность смартфонов и «бум» в разработке мобильных приложений) и поэтому решили изучать программирование именно посредством этого языка программирования. И теряют время, которого очень мало перед экзаменом. К сожалению, такой подход нередко приводит к разочарованиям – Java является строгим и каноничным языком программирования, изучение которого требует не только прочных знаний в математике, но и определенного способа мышления. Где ж его взять, спросите вы. Ответ будет прост: он формируется сам, по мере приобретения программистского опыта. Именно поэтому Java – не лучший выбор в качестве первого языка. Как не наступить на эти грабли и выбрать «правильный» язык программирования?

Во-первых, оцените уровень начальных знаний. Даже школьный опыт программирования на PascalABC – это лучше, чем ничего;

Во-вторых, огромное значение имеет возраст ученика.

В-третьих, проанализируйте объективное значение языка программирования : его востребованность в обучении и на рынке труда, перспективы развития

Ошибка №2. Не читать специализированных книг. Знаете, нам бы очень хотелось, чтобы язык программирования изучался легко и непринужденно – пришел на занятие, послушал теорию, прорешал все задачки и, довольный, ушел домой. К сожалению, так профессию не постигнуть и к ЕГЭ не подготовиться. Изучению программирования требуется уделять много времени, а не просто режим 1раз в неделю по 1 академическому часу. А для этого следует читать книги (благо, их сейчас немало), статьи, форумы. Другими словами, нужно быть «в теме». Особенно это касается тех, кто изучает программирование самостоятельно. Зачастую начинающие кодеры читают книги по специальности как художественную литературу – глава за главой. Замечу, что книги для общего развития тоже существуют, но большинство учебной литературы все же требует вдумчивого чтения и, что особенно важно, практики. Нужно решать все задачи и упражнения, которые приводятся в учебниках, придумывать себе задания, разбирать код из открытых источников.

Ошибка №3. Не изучать английский. По опросу ВЦИОМ, проведенному в 2014 году, 95% респондентов считают, что знание иностранного языка повышает их умственные способности. Разумеется, наибольшей популярностью пользуется англий-

ский. Это к вопросу общего развития. Теперь перейдем от общего к частному – как знание английского влияет на успехи в изучении программирования? Здесь все просто: тот школьник, который хоть немного разбирается в лексике и грамматике иностранного (читай – английского) языка, быстрее понимает и лучше усваивает синтаксис и строение изучаемого языка программирования. Простой пример: вы знаете, как переводится слово break? Тогда без труда догадаетесь, что эта команда означает, если встречается в циклах, без которых не обходится ни один язык программирования. Все верно – она его прерывает (break с английского так и переводится – «прервать», «прекратить»).

Пример цикла на javascript, который перебирает элементы от 1 до 99 и прерывается, если текущий элемент равен 15

Ошибка №4. Костыли в коде. Одна из самых частых фраз, которую слышат преподаватели на занятиях по программированию, звучит так: «Работает же!». Большинство начинающих программистов считает, что главное, чтобы твой код скомпилировался и заработал, а уж как он написан – дело второстепенное. Здесь напрашивается простое сравнение представьте себе телегу. Обычную деревянную телегу. Она имеет 4 колеса, может передвигаться и ею даже можно управлять. Но назвать ее транспортным средством как-то не получается, и на наших дорогах ее не встретишь. Все мы предпочитаем перемещаться на более комфортном транспорте, не правда ли? Вы бы, например, хотели получить в подарок не машину, а телегу? Она ведь тоже в известном смысле «работает»! Плохой код можно сравнить с телегой – вроде бы “работает”, но ездить на ней не хочется Так же и с изучением языка программирования – код начинающих программистов буквально пестрит неочевидными ходами, неуклюжими решениями, громоздкими структурами, словом, всем тем, что опытные разработчики и преподаватели называют «костылями». Прописная истина – код должен быть чистым, понятным и логичным. Оно должен быть оптимизирован, избавлен от неочевидных решений, повторяющихся фрагментов, избыточных записей и прочей «шелухи».

Ошибка №5. Нежелание развиваться. Обучение – это процесс двусторонний, и ни один преподаватель не сможет вложить в голову ученику знания, если у последнего не будет мотивации и настоящего интереса к программированию. Учитель лишь может подогревать этот интерес, направлять на правильный путь, но 50% успеха зависит от самого ученика. Нужно самостоятельно пытаться понять, как устроен язык программирования, решать практические задачи «до победного конца», искать ответы на возникающие вопросы (для этого, кстати, существует официальная документация к языка программирования, которую большинство студентов игнорирует), интересоваться не только изучаемой технологией, но и смежными дисциплинами… Простыми словами, нужно учиться учиться. И тогда – все получится.

А теперь более конкретно по задачам ЕГЭ

Номер задания	**Проверяемые элементы содержания**	**Уровень сложности** **Б базавый** **П повышенный** **В высокий**
8	Знание основных конструкций языка программирования, понятия переменной, оператора присваивания	Б
11	Умение исполнить рекурсивный алгоритм	Б
19	Работа с массивами (заполнение, считывание, поиск, сортировка, и др.)	П
20	Анализ программ с циклами и условными операторами Анализ алгоритма, содержащего вспомогательные алгоритмы, цикл и ветвление	П
21	Умение анализировать программу, использующую про-	П

	цедуры и функции Анализ программ с циклами и подпрограммами	
22	Умение анализировать результат исполнения алгоритма	П
24	Умение прочесть фрагмент программы на языке программирования и исправить допущенные ошибки	П
25	Умения написать короткую (10-15 строк) простую программу (например, обработки массива) на языке программирования или записать алгоритм на естественном языке	В
27	Умения создавать собственные программы (30-50 строк) для решения задач средней сложности	В

Задача 8. (Задачи, представленные в примерах, взяты с сайта К. Ю. Полякова с личного разрешения автора)

Знание основных конструкций языка программирования, понятия переменной, оператора присваивания. ***Что нужно знать для решения:***

1. основные конструкции языка программирования:
2. объявление переменных
3. оператор присваивания
4. оператор вывода
5. циклы
6. уметь выполнять ручную прокрутку программы
7. уметь выделять переменную цикла, от изменения которой зависит количество шагов цикла
8. уметь определять количество шагов цикла
9. уметь определять переменную, которая выводится на экран

Ошибки бывают от невнимательности, путают переменные, которые выводятся на монитор (внимательно смотрим на оператор вывода), путают переменную –счетчик и переменную, от изменения которой зависит количество шагов цикла.

- Если количество повторов в цикле очень большое, то ручная трассировка, скорее всего, приведет к вычислительной ошибке.
- Часто, если начальные значения переменных (например, **s** и **k)** не равны нулю, это не замечают, а считают их обнулёнными и возникает ошибка по невнимательности.
- Необходимо помнить, что количество членов арифметической прогрессии на 1 больше, чем количество шагов, которые необходимы для перехода от первого значения к последнему

Для быстрого и успешного выполнения рассмотренного задания важно было не механически выполнить алгоритм, а понять закономерность, которую он выражает, и, воспользовавшись ей, найти решение.

Важное замечание:

Практически во всех заданиях на исполнение алгоритма можно избежать большого объема рутинной работы, выявив закономерность, реализуемую алгоритмом.

Пример 8.1Определите, что будет напечатано в результате работы следующего фрагмента программы:

```
var n, s: integer;
begin
  n := 4;
  s := 15;
  while s <= 250 do begin
    s := s + 12;
```

```
    n := n + 2
  end;
  write(n)
end.
```

Как мы решаем: 1).Переменная s изменится (250-15)/12=19 раз. Значит, n:=2*19+4=42, и делаем ошибку! Цикл повторится 20 раз (19+1) и верный ответ 2*20+4= 44! Не забываем, что (количество членов арифметической прогрессии на 1 больше, чем количество шагов, которые необходимы для перехода от первого значения к последнему)!

А в циклах с параметром (со счетчиком) необходимо уметь считать количество повторений по формуле для цикла вида for k := n to m do количество повторений равно n – m + 1 (for k := n downto m do количество повторений равно m – n +1)

```
s := 0;
for k := 4 to 7 do
s := s + 8;
writeln(s);
End.
```

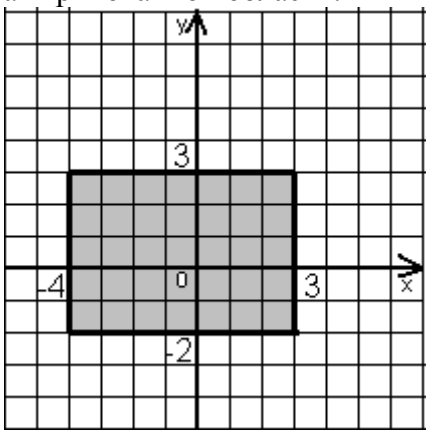
Пояснение.

Цикл «for k := 4 to 7 do» выполняется четыре раза (7 – 4 +1). Каждый раз переменная s увеличивается на 8. Поскольку изначально s = 0, после выполнения программы получим: s = 8 · 4 = 32.

Проанализировав результаты ЕГЭ, видим, что наши выпускники неплохо справляются с задачей 8. Практически, эти же знания нужны и для 24 задачи, но здесь результат плачевнее. Всегда говорю своим ребятам: «Не надо её боятся, надо просто решить!. Она уже решена, сделай трассировку, как и в 8 и всё увидишь! »

Разберем задачу (24 ЕГЭ).

Задание 1Составить программу, по которой определяется, лежит ли точка с заданными координатами (x, y) внутри заштрихованной области.



Составим условия, которым должны отвечать точки, попавшие в выделенную область (прямоугольник).

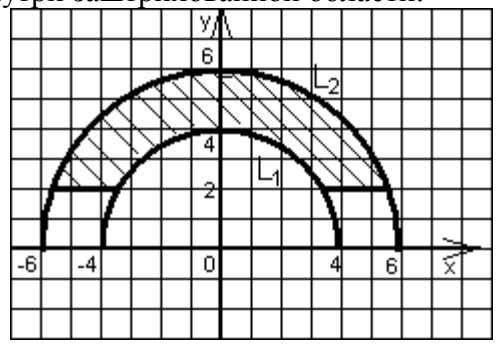
Во первых, заштрихованная область ограничена по координате x, она находится правее x=-4 и левее x=3, следовательно, координата x должна удовлетворять условию: (x>=-4) and (x<=3).

Во вторых, заштрихованная область ограничена по координате y, она находится ниже y=3 и выше y=-2, следовательно, координата y должна удовлетворять условию: (y>=-2) and (y<=3).

Так как эти два условия должны выполняться одновременно, то запишем условный оператор (фрагмент программы):

if (x>=-4) and (x<=3) and (y>=-2) and (y<=3) **then writeln(**‘точка принадлежит’**)**
else writeln(‘точка не принадлежит’**);**

Задание 2. Составить программу, по которой определяется, лежит ли точка с заданными координатами (x, y) внутри заштрихованной области.

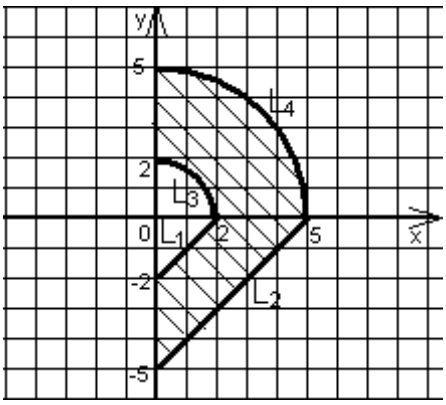


Заштрихованная область представляет собой часть кольца, ограниченного прямой y=2. Кольцо составлено двумя концентрическими окружностями с центром в начале координат. Запишем уравнения этих окружностей:

L_1: $x^2+y^2=4^2$
L_2: $x^2+y^2=6^2$

Так как точка должна лежать вне окружности L_1 и одновременно внутри окружности L_2, то получим условия: $x^2+y^2>=16$ и $x^2+y^2<=36$.
Запишем условный оператор (фрагмент программы):
if ($x^2+y^2>=16$) and ($x^2+y^2<=36$) and (y>=2) **then writeln(**‘точка принадлежит’**)**
else writeln(‘точка не принадлежит’**);**

Задание 3. Составить программу, по которой определяется, лежит ли точка с заданными координатами (x, y) внутри заштрихованной области.

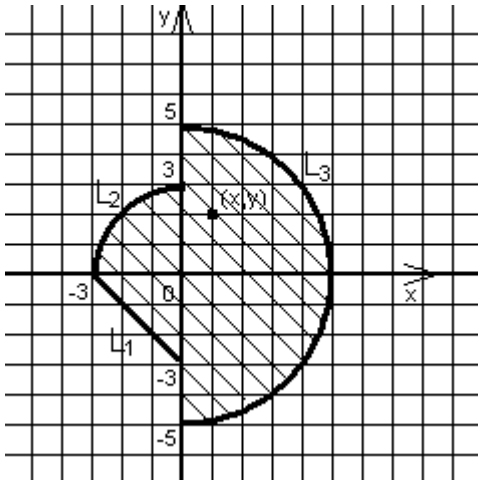


L_1: y=x-2
L_2: y=x-5
L_3: $x^2+y^2=4$
L_4: $x^2+y^2=25$
1 часть: если y<=0 и x>=0, то y>=x-5 и y<=x-2
2 часть: если y>0 и x>=0, то $x^2+y^2>=4$ и $x^2+y^2<=25$

Так как точка не может попасть одновременно в 1 и 2 часть, то между условиями 1 и 2 части должен быть союз ИЛИ

if (y<=0) and (x>=0) and (y>=x-5) and (y<=x-2) **or** (y>0) and (x>=0) and (x*x+y*y>=4) and (x*x+y*y<=25)
then writeln(‘точка принадлежит’**)**
else writeln(‘точка не принадлежит’**);**

Задание 4. Составить программу, по которой определяется, лежит ли точка с заданными координатами (x, y) внутри заштрихованной области.

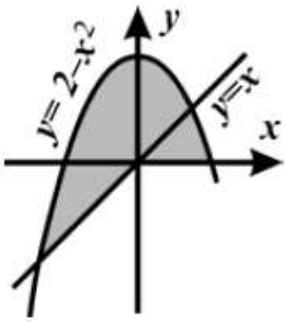


L_1: y=-x-3
L_2: $x^2+y^2=3^2$
L_3: $x^2+y^2=5^2$
1 часть: если x<=0, то $x^2+y^2<=9$ и y>=-x-3
2 часть: если x>=0, то x^2+y2<=25

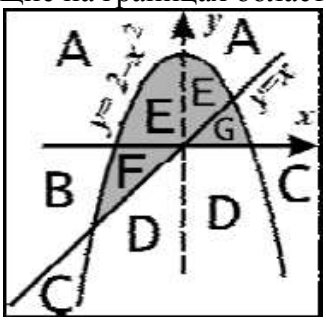
if (x<=0) and (x*x+y*y<=9) and (y>=-x-3) or (x>=0) and (x*x+y*y<=25) **then**
writeln(‘точка принадлежит’**)**
else writeln(‘точка не принадлежит’**);**

Задание 5. Требовалось написать программу, которая вводит с клавиатуры координаты точки на плоскости (x, y – действительные числа) и определяет принадлежность точки заштрихованной области, включая ее границы. Программист торопился и написал программу неправильно. Вот она:

```
var x,y: real;
begin
readln(x,y);
if y>=x then
if y>=0 then
if y<=2-x*x then
write('принадлежит')
else
write('не принадлежит')
end.
```



Последовательно выполните следующее.
1. Перерисуйте и заполните таблицу, которая показывает, как работает программа при аргументах, принадлежащих различным областям (A, B, C, D, E, F и G). Точки, лежащие на границах областей, отдельно не рассматривать.



Область	Условие 1 (y>=x)	Условие 2 (y>=0)	Условие 3 (y<=2–x*x)	Программа выведет	Область обрабатывается верно
A					
B					
C					
D					
E					
F					
G					

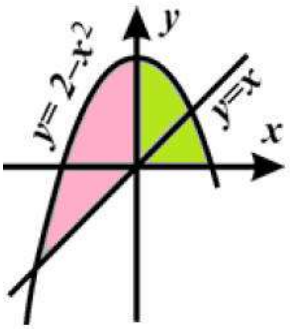
В столбцах условий укажите "да", если условие выполнится, "нет" если условие не выполнится, "—" (прочерк), если условие не будет проверяться, «не изв.», если программа ведет себя по-разному для разных значений, принадлежащих данной области. В столбце "Программа выведет" укажите, что программа выведет на экран. Если программа ничего не выводит, напишите "—" (прочерк). Если для разных значений, принадлежащих области, будут выведены разные тексты, напишите «не изв». В последнем столбце укажите "да" или "нет".

2. Укажите, как нужно доработать программу, чтобы не было случаев ее неправильной работы. (Это можно сделать несколькими способами, достаточно указать любой способ доработки исходной программы.)

Решение.

Область	Условие 1 (y>=x)	Условие 2 (y>=0)	Условие 3 (y<=2–x*x)	Программа выведет	Область обрабатывается верно
A	да	да	нет	не принадлежит	да
B	да	нет	—	—	нет
C	нет	—	—	—	нет
D	нет	—	—	—	нет
E	да	да	да	принадлежит	да
F	да	нет	—	—	нет
G	нет	—	—	—	нет

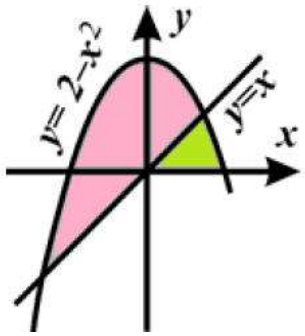
Для составления правильной программы можно разбить область на две части:



Тогда программа будет выглядеть следующим образом:

if (x<0) and (y<=2-x*x) and (y>=x) or (x>=0) and (y>=0) and (y<=2-x*x) **then**

write('принадлежит')
else write('не принадлежит');
2 способ - разбиение на две других части:



Тогда программа будет выглядеть следующим образом:
if (y<=2-x*x) and (y>=x) or (y<=x) and (y<=2-x*x) and (x>=0) **then write**('принадлежит')
else write('не принадлежит');

Типичные ошибки учащихся, допущенные при выполнении задания 24:

- неверно указана точка, для которой программа работает неправильно;
- указано несколько точек, некоторые из них – неверно;
- исправлена только одна из ошибок в программе;
- неправильно описана заштрихованная область;
- приведен фрагмент программы и не указано, в какое место программы его необходимо вставить. С 2017 года задания такого типа не включают в варианты ЕГЭ

Задача 24(2)

Факториалом натурального числа N (обозначается $N!$) называется произведение всех натуральных чисел от 1 до N. Например, $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$. Даны целые положительные числа A и B, $0 < A \le B$. Необходимо найти количество таких натуральных K, для которых $A \le K! \le B$. Например, при $A = 5$, $B = 25$ ответ должен быть равен 2 (подходящие значения K — 3 и 4, их факториалы 6 и 24 попадают в заданный интервал). Для решения этой задачи ученик написал программу, но, к сожалению, его программа неправильная.

```
var a, b, k, f, d: integer;
begin
    read(a, b);
    f := 1;
    k := 1;
    d := 1;
    while f <= b do begin
        if f > a then
            d := d + 1;
        k := k + 1;
        f := f * k
    end;
    writeln(d)
end.
```

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе чисел 5 и 25.
2. Приведите пример входных данных, при которых программа выведет верный ответ.

3. Найдите в программе все ошибки (известно, что их не больше двух) и исправьте их. Для каждой ошибки выпишите строку, в которой она допущена, и приведите эту же строку в исправленном виде.

Достаточно указать ошибки и способ их исправления для одного языка программирования.

Обратите внимание: Вам нужно исправить приведённую программу, а не написать свою. Вы можете только заменять ошибочные строки, но не можете удалять строки или добавлять новые. Заменять следует только ошибочные строки: за исправления, внесённые в строки, не содержащие ошибок, баллы будут снижаться.

Решение.

1. При вводе чисел 5 и 25 программа выведет число 3.

2. Верный ответ получается, если первое число является точным факториалом. Возможные пары: 2 и 5, 6 и 24. Ученику достаточно привести одну конкретную пару чисел.

3. Программа содержит две ошибки.

1) Неверная инициализация счётчика *D*. Из-за этого ответ оказывается на единицу больше, чем нужно.

2) Ошибочное сравнение. Вместо нестрогого сравнения *F* и *A* выполняется строгое. Из-за этого в тех случаях, когда *A* оказывается точным факториалом, ответ получается на единицу меньше.

Пример исправления для языка Паскаль:

Первая ошибка:

d := 1;

Исправленная строка:

d := 0;

Вторая ошибка:

if f > a then

Исправленная строка:

if f >= a then

В данных задачах необходимо уметь анализировать программу с полным и неполным ветвлением.

Задача 11. Рассматривается реурсивная подпрограмма, процедура или функция.

Трудности в задании 11.

(базовый уровень, время – 5 мин) (все задачи взяты с сайта К. Ю. Полякова с личного разрешения автора)

Тема: рекурсивные алгоритмы.

Что нужно знать:

- рекурсия – это приём, позволяющий свести исходную задачу к одной или нескольким более простым задачам того же типа
- чтобы определить рекурсию, нужно задать
  - условие остановки рекурсии (базовый случай или несколько базовых случаев)
  - рекуррентную формулу
- любую рекурсивную процедуру можно запрограммировать с помощью цикла
- рекурсия позволяет заменить цикл и в некоторых сложных задачах делает решение более понятным, хотя часто менее эффективным
- существуют языки программирования, в которых рекурсия используется как один из основных приемов обработки данных (Lisp, Haskell)

Готовясь к сдаче ЕГЭ, можно рассмотреть задачи из разных вариантов за разные годы, в которых просматривается рекурсия: на определение значений функции; на

нахождение количества программ, преобразующих одно число в другое; задачи про цепочки символов.

Можно попробовать их запрограммировать, чтобы лучше понять, как это работает. Многие из этих примеров можно более эффективно решить, не применяя рекурсии. Конечно же, на экзамене такие задачи делаются письменно, без компьютерных вычислений. Однако, предложив решать такие задачи своим ученикам на компьютере, я порадовалась тому, что они стали лучше понимать ход решения таких задач, а заодно и освоили рекурсию.

Пример задания:

Пример 11.1 Ниже записана рекурсивная процедура:

```
procedure F(n: integer);
begin
  if n > 1 then begin
    F(n - 4);
    write(n);
    F(n div 2);
  end;
end;
```

Что будет напечатано на экране при выполнении вызова F(11)?

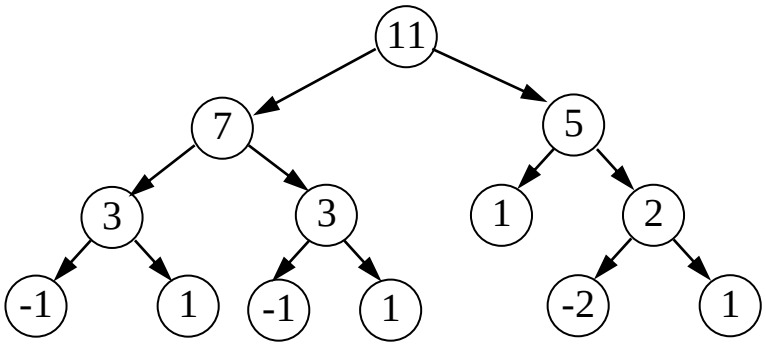
Решение (подстановка значений):

1) при вызове F(11) условие n>1 выполняется, программа входит в тело условного оператора
2) вызывается F(n-4) = F(7), затем выводится 11 и вызывается F(n div 2) = F(5); все это условно запишем так:
   F(11)= F(7) 11 F(5)
3) аналогично рассматриваем F(7) и F(5), учитывая, что при n <= 1 процедура вообще ничего не выводит на экран:
   F(7)= F(3) 7 F(3)
   F(5)= F(1) 5 F(2) = 5 F(2)
4) теперь неизвестны F(3), F(2) (как мы уже говорили, F(1) ничего не выводит), разбираем их:
   F(3)= F(-1) 3 F(1) = 3
   F(2)= F(-2) 2 F(0) = 2
5) собираем все обратно, удаляя пробелы:
   F(5)= 5 F(2) = 52
   F(7)= F(3) 7 F(3) = 373
   F(11)= F(7) 11 F(5) = 3731152

6) Ответ: **3731152.**

Решение (дерево и таблица, Л. А. Тумарина, г. Электросталь):

1) Для упрощения дальнейшего решения построим дерево вызовов, пока не учитывая порядок вывода чисел на экран:



2) При **n<=1** программа ничего не выводит:

n	Что вызывается	Что выведет программа
<=1	–	–

3) При **n=2** программа вызовет **F(–2)**, затем напечатает **2**, затем вызовет **F(1)**:

n	Что вызывается	Что выведет программа
<=1	–	–
2	**F(–2);** **write(n);** **F(1);**	– 2 –

4) При **n=3** программа вызовет **F(–1)**, затем напечатает **3**, затем вызовет **F(1)**:

n	Что вызывается	Что выведет программа
<=1	–	–
2	**F(–2);** **write(n);** **F(1);**	– 2 –
3	**F(–1);** **write(n);** **F(1);**	– 3 –

5) При **n=5** программа вызовет **F(1)**, затем напечатает **5**, затем вызовет **F(2)**. А вызов **F(2)** напечатает **2** (см. строку 2 таблицы). Учитываем, что программа выводит все числа подряд без пробелов:

n	Что вызывается	Что выведет программа
<=1	–	–

2	`F(-2);`
	`write(n);`
	`F(1);`
		–
		2
		–

3	`F(-1);`
	`write(n);`
	`F(1);`
		–
		3
		–

5	`F(1);`
	`write(n);`
	`F(2);`
		–
		5
		2

6) При **n=7** программа вызовет **F(3)**, затем напечатает **7**, затем вызовет **F(3)**. А вызов **F(3)** напечатает **3** (см. строку 3 таблицы):

n	Что вызывается	Что выведет программа
<=1	–	–

2	`F(-2);`
	`write(n);`
	`F(1);`
		–
		2
		–

3	`F(-1);`
	`write(n);`
	`F(1);`
		–
		3
		–

5	`F(1);`	–

	`write(n);` `F(2);`	5 2

7	`F(3);`
	`write(n);`
	`F(3);`
		3
		7
		3

7) При **n=11** программа вызовет **F(7)**, затем напечатает **11**, затем вызовет **F(5)** (см. соответствующие строки таблицы):

n	**Что вызывается**	**Что выведет программа**
<=1	`-`	–

2	`F(-2);`
	`write(n);`
	`F(1);`
		–
		2
		–

3	`F(-1);`
	`write(n);`
	`F(1);`
		–
		3
		–

5	`F(1);`
	`write(n);`
	`F(2);`
		–
		5
		2

7	**F(3);** **write(n);** **F(3);**	3 7 3
11	**F(7);** **write(n);** **F(5);**	373 11 52

8) Ответ: **3731152**.

. Пример 11.2 Ниже записаны две рекурсивные процедуры: F и G:

```
procedure F(n: integer); forward;
procedure G(n: integer); forward;
procedure F(n: integer);
begin
if n > 0 then
  G(n - 1);
end;
procedure G(n: integer);
begin
writeln('*');
if n > 1 then
   F(n - 2);
end;
```

Сколько символов «звёздочка» будет напечатано на экране при выполнении вызова F(11)?

Решение:

1) заметим, что каждая функция вызывает другую (это называется косвенная рекурсия), причём только один раз
2) вот цепочка вызовов:

   $F(11) \to G(10) \to F(8) \to G(7) \to F(5) \to G(4) \to F(2) \to G(1)$
3) за один вызов функции G выводится одна звёздочка, внутри функции F звездочки не выводятся, поэтому за 4 вызова G будет выведено 4 звездочки

Ответ: 4.

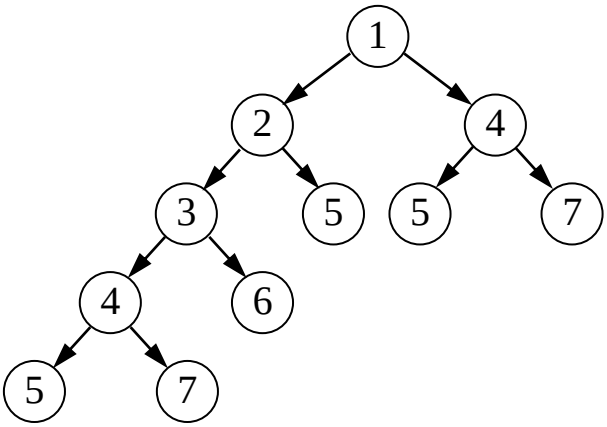
Пример 11.1. Дан рекурсивный алгоритм:

```
procedure F(n: integer);
begin
  writeln(n);
  if n < 5 then begin
    F(n + 1);
    F(n + 3)
  end
end;
```

Найдите сумму чисел, которые будут выведены при вызове F(1).

Решение (вариант 1, построение дерева вызовов):

1) поскольку в начале каждого вызова на экран выводится значение единственного параметра функции, достаточно определить порядок рекурсивных вызовов и сложить значения параметров
2) поскольку при $n < 5$ выполняется два рекурсивных вызова, решать такую задачу «на бумажке» удобно в виде двоичного дерева (в узлах записаны значения параметров при вызове функции):



3) складывая все эти числа, получаем 49
4) ответ: 49.

Решение (вариант 2, подстановка):

1) можно обойтись и без дерева, учитывая, что при каждом вызове с n < 5 происходит два рекурсивных вызова; сумму чисел, полученных при вызове $F(n)$, обозначим через $S(n)$:

$$S(n) = \begin{cases} n + S(n+1) + S(n+3), & n < 5 \\ n, & n \geq 5 \end{cases}$$

2) выполняем вычисления:

$S(1) = 1 + S(2) + S(4)$

$S(2) = 2 + S(3) + S(5) = 7 + S(3)$

$S(3) = 3 + S(4) + S(6) = 9 + S(4)$

$S(4) = 4 + S(5) + S(7) = 16$

3) теперь остаётся вычислить ответ «обратным ходом»:

$S(3) = 9 + 16 = 25$

4) $S(2) = 7 + 25 = 32$

$S(1) = 1 + 32 + 16 = 49$

Ответ: 49.

Пример 11.3. Дан рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 if n < 6 then begin
   F(n+2);
   F(n*3)
 end
end;
```

Найдите сумму чисел, которые будут выведены при вызове F(1).

Решение (вариант 1, метод подстановки):

1) сначала определим рекуррентную формулу; обозначим через G(n) сумму чисел, которая выводится при вызове F(n)
2) при n >= 6 процедура выводит число n и заканчивает работу без рекурсивных вызовов:
   G(n) = n при n >= 6
3) при n < 6 процедура выводит число n и дважды вызывает сама себя:
   G(n) = n + G(n+2) + G(3n) при n < 6
4) в результате вызова F(1) получаем
   G(1) = 1 + G(3) + G(3)
   G(3) = 3 + G(5) + G(9) = 3 + G(5) + 9
   G(5) = 5 + G(7) + G(15) = 5 + 7 + 15 = 27
5) используем обратную подстановку:
   G(3) = 3 + G(5) + 9 = 3 + 27 + 9 = 39
   G(1) = 1 + 2*G(3) = 79
6) Ответ: 79.

Решение (вариант 2, динамическое программирование):

1) п. 1-3 такие же, как в первом варианте решения
2) заполняем таблицу G(n) при n >= 6 (где G(n) = n)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
G(n)						6	7	8	9	10	11	12	13	14	15

3) остальные ячейки заполняем, начиная с n = 5 справа налево, используя формулу :
   G(n) = n + G(n+2) + G(3n)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
G(n)	79	30	39	22	27	6	7	8	9	10	11	12	13	14	15

4) ответ читаем в самой левой ячейке

Ответ: 79.

Пример 11.4. Дан рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln('*');
 if n > 0 then begin
   F(n-2);
   F(n div 2)
 end
end;
```

Сколько символов "звездочка" будет напечатано на экране при выполнении вызова F(7)?

Решение (вариант 1, составление полной таблицы):

1) сначала определим рекуррентную формулу; обозначим через G(n) количество звездочек, которые выводит программа при вызове F(n)
2) из программы видим, что
   G(n) = 1 при всех n <= 0
   G(n) = 1 + G(n-2) + G(n div 2) при n > 0
3) вспомним, что **n div 2** – это частное от деления n на 2
4) по этим формулам заполняем таблицу, начиная с нуля:
   G(0) = 1
   G(1) = 1 + G(-1) + G(0) = 1 + 1 + 1 = 3
   G(2) = 1 + G(0) + G(1) = 1 + 1 + 3 = 5
   G(3) = 1 + G(1) + G(1) = 1 + 3 + 3 = 7
   G(4) = 1 + G(2) + G(2) = 1 + 5 + 5 = 11
   G(5) = 1 + G(3) + G(2) = 1 + 7 + 5 = 13
   G(6) = 1 + G(4) + G(3) = 1 + 11 + 7 = 19
   G(7) = 1 + G(5) + G(3) = 1 + 13 + 7 = 21

n	0	1	2	3	4	5	6	7
G(n)	1	3	5	7	11	13	19	21

Ответ: 21.

Решение (вариант 2, «с конца»):
1) пп. 1-3 – как в варианте 1
2) по формулам G(7) = 1+ G(5) + G(3), поэтому нужно найти G(5) и G(3)
3) G(5) = 1 + G(3) + G(2), нужны G(3) и G(2)
4) G(3) = 1 + G(1) + G(1), нужно G(1)
5) G(2) = 1 + G(0) + G(1) = 2 + G(1), нужно G(1)
6) G(1) = 1 + G(-1) + G(0) = 1 + 1 + 1 = 3
7) теперь идем «обратным ходом»:
   G(2) = 2 + G(1) = 5
   G(3) = 1 + G(1) + G(1) = 1 + 3 + 3 = 7
   G(5) = 1 + G(3) + G(2) = 1 + 7 + 5 = 13
   G(7) = 1 + G(5) + G(3) = 1 + 13 + 7 = 21

Ответ: 21.

. Пример 11.5 *Алгоритм вычисления значений функций F(n) и G(n), где n – натуральное число, задан следующими соотношениями:*

F(1) = 1; G(1) = 1;
F(n) = F(n – 1) – G(n – 1),
G(n) = F(n–1) + G(n – 1), при n >=2

Чему равно значение величины F(5)/G(5)?
В ответе запишите только целое число.

Решение:
1) фактически рекуррентная формула задана для пары (F(n); G(n))
2) замечаем, что F(n) – это разность предыдущей пары, а G(n) – сумма тех же значений
3) заполняем таблицу, начиная с известной первой пары

n	1	2	3	4	5
F(n)	1	0	–2	–4	–4
G(n)	1	2	2	0	–4

4) искомое значение F(5)/G(5) равно 1
   Ответ: 1.

Пример 11.6 *Алгоритм вычисления значения функции F(n), где n – натуральное число,*
задан следующими соотношениями:

F(1) = 1
F(n) = F(n–1) * n, при n > 1

Чему равно значение функции F(5)?
В ответе запишите только целое число.

Решение:

1) используя заданную рекуррентную формулу, находим, что
   F(5) = F(4) * 5
2) применив формулу еще несколько раз, получаем
   F(5) = F(3) * 4 * 5 = F(2) * 3 * 4 * 5 = F(1) * 2 * 3 * 4 * 5
3) мы дошли до базового случая, который останавливает рекурсию, так как определяет значение F(1) = 1
4) окончательно F(5) = 1 * 2 * 3 * 4 * 5 = 120

Ответ: 120.

Пример 11.7 *Процедура F(n), где n – натуральное число, задана следующим образом (язык Паскаль):*

```
procedure F(n: integer);
begin
  if n < 3 then
    write('*')
  else begin
    F(n-1);
    F(n-2);
    F(n-2)
  end;
end;
```

Сколько звездочек напечатает эта процедура при вызове F(6)? В ответе запишите только целое число.

Решение:

1) эта задача по сути такая же, как и предыдущая, но «завёрнута» в другой фантик: для **n < 3** (то есть, для 1 и 2) функция выводит одну звездочку
   F(1) = F(2) = 1
   а для бóльших n имеем рекуррентную формулу
   F(n) = F(n-1) + F(n-2) + F(n-2)
   = F(n-1) + 2*F(n-2)
2) запишем в таблицу базовые случаи

n	1	2	3	4	5	6
F(n)	1	1				

3) заполняем таблицу, используя рекуррентную формулу:

n	1	2	3	4	5	6
F(n)	1	1	3	5	11	21

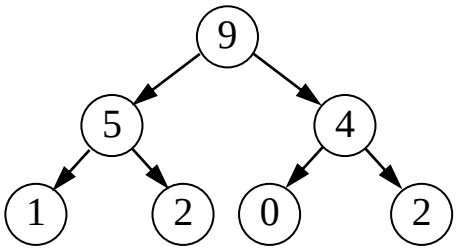
   F(3) = F(2) + 2*F(1) = 3
   F(4) = F(3) + 2*F(2) = 5
   F(5) = F(4) + 2*F(3) = 11
   F(6) = F(5) + 2*F(4) = 21

Ответ: 21.

Пример 11.8 Туть суть решения почти та же: уходим в "глубину" рекурсии с одним изменением: вывод только один раз и после условия.

Что напечатает программа:

```
      procedure F(n:integer);
begin
  if n>3 then begin
   F(n-4);
    F(n div 2);
   end;
 write(n);  end;
begin
  F(9);
end
```

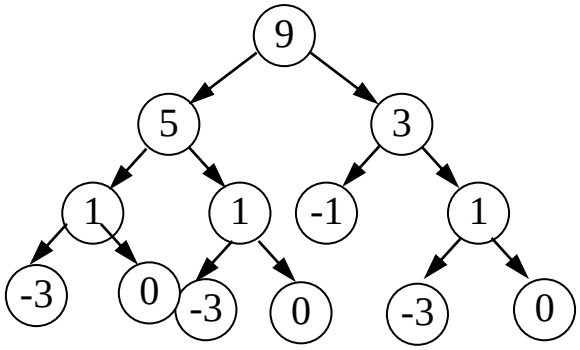


Ответ: 1250249

Пример 11.9

Что напечатает программа:

```
program n_10;
  procedure F(n: integer);
   begin
    if (n>0) then begin
   F(n-4);
      writeln(n);
     F(n div 3)
   end;
  end;
begin
 F(9);
end.
```



Решение:

1. Сначала заходим в функцию с параметром 9

2. Далее с параметром 5

3. С параметром 1:

3.1. Далее происходит передача параметра -3, что посылает нас назад (точнее, ничего не происходит, т.к. условие не выполняется). Возвращаемся в предыдущую (номер 3).

3.2. !!!Выводим 1!!!

3.3. Передаём 0, что тоже не удовлетворяет условию, возвращаемся назад (в номер 3).

3.4. Возвращаемся назад (номер 2).

2.1. !!!Выводим 5!!!

2.2. Заходим в процедуру с параметром 1 (5 делим нацело на 3 = 1).

2.2.1. Передаём в качестве параметра -3 и возвращаемся назад.

2.2.2. !!!Выводим 1!!!

2.2.3. Передаём 0, что тоже не удовлетворяет условию, возвращаемся назад.

2.2.4. Возвращаемся в номер 2.

2.3. Возвращаемся в номер 1.

1.1. !!!Выводим 9!!!

1.2. Передаём в качестве параметра 3 (9 делить нацело на 3 = 3).

1.2.1. Передаём в качестве параметра -1 и возвращаемся назад.

1.2.2. !!!Выводим 3!!!

1.2.3. Передаём в качестве параметра 1 (3 делить нацело на 3 = 1).

1.2.3.1. Передаём в качестве параметра -3 и возвращаемся назад.

1.2.3.2. !!!Выводим 1!!!

1.2.3.3. Передаём 0, что тоже не удовлетворяет условию, возвращаемся назад.

1.3. Возвращаемся назад в 1.

4. Выходим из рекурсии и завершаем работу.

Таким образом выводится следующее:**Ответ: 151931**

Пример 11.10

Запишите подряд без пробелов и разделителей все числа, которые будут напечатаны на экране при выполнении вызова **F(4).** Числа должны быть записаны в том же порядке, в котором они выводятся на экран.

```
procedure F(n: integer);
begin
    if n > 0 then begin
        F(n - 1);
        writeln(n);
        F(n - 2)
    end
end;
```

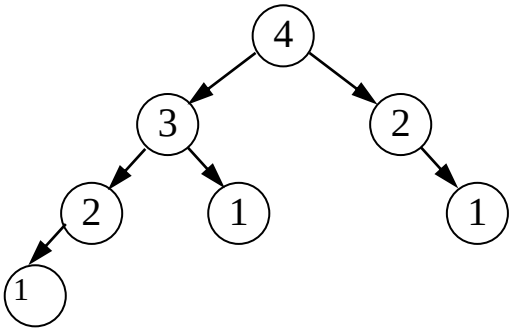
Решение:

Первым действием процедура F(4) вызовет процедуру F(3), которая вызовет процедуру F(2). После этого процедура F(2) вызовет процедуру F(1), которая выведет на экран 1. После этого процедура F(2) выведет на экран 2. Далее процедура F(3) следующим шагом своего алгоритма выведет на экран число 3, после чего будет вызвана процедура F(1), которая выведет на экран 1.

Далее процедура F(4) следующим шагом своего алгоритма выведет на экран 4 и вызовет процедуру F(2), которая вызовет процедуру F(1), после чего на экран будет выведена 1. Далее процедура F(2) выведет на экран 2.

Ответ: 1231412.

Источник: Демонстрационная версия ЕГЭ – 2019 по информатике.



Задание19 (повышенный уровень, время – 5 мин)

Тема: Работа с массивами и матрицами в языке программирования.

Что нужно знать:

- работу цикла **for** (цикла с переменной)
- массив – это набор однотипных элементов, имеющих общее имя и расположенных в памяти рядом
- для обращения к элементу массива используют квадратные скобки, запись **A[i]** обозначает элемент массива **A** с номером (индексом) **i**
- описание одномерного массива: var имя массива: array [тип индекса] of базовый тип;
- матрица (двухмерный массив) – это прямоугольная таблица однотипных элементов
- если матрица имеет имя A, то обращение **A[i,k]** обозначает элемент, расположенный на пересечении строки **i** и столбца **k**
- элементы, у которых номера строки и столбца совпадают, расположены на главной диагонали

A[1,1]			
	A[2,2]		
		A[3,3]	
			A[4,4]

- выше главной диагонали расположены элементы, у которых номер строки **меньше** номера столбца:

	A[1,2]	A[1,3]	A[1,4]
		A[2,3]	A[2,4]
			A[3,4]

- ниже главной диагонали расположены элементы, у которых номер строки **больше** номера столбца:

A[2,1]			
A[3,1]	A[3,2]		
A[4,1]	A[4,2]	A[4,3]	

Сложность этих задач в том, что все действия нужно «прокручивать в уме» (или на бумаге), не используя компьютер для отладки;
В двумерном массиве главная проблема – не перепутать столбцы и строки; номер строки – это (по соглашению) первый индекс элемента матрицы, а номер столбца – второй

Пример 19.1В программе описан одномерный целочисленный массив с индексами от 0 до 10. Ниже представлен фрагмент программы, обрабатывающей данный массив:

```
s:=0;
n:=10;
for i:=0 to n-2 do begin
  s:=s+A[i]-A[i+2]
end;
```

В начале выполнения этого фрагмента в массиве находились трёхзначные натуральные числа. Какое наибольшее значение может иметь переменная s после выполнения данной программы?
Решение:

сначала попытаемся понять, что же делает эта программа; возьмем массив из пяти элементов (n = 4):

0	1	2	3	4
A[0]	A[1]	A[2]	A[3]	A[4]

1) переменная s будет изменяться следующим образом:
```
s := 0
s := s + A[0] – A[2]
s := s + A[1] – A[3]
s := s + A[2] – A[4]
```
2) в итоге после всех действий
```
s:= A[0]–A[2]+A[1]–A[3]+A[2]–A[4] = A[0] + A[1] – A[3] – A[4]
```
3) это значит, что значение **s** всегда будет равно сумме двух первых элементов массива минус сумма двух последних элементов
4) все числа – трёхзначные, то есть принадлежат отрезку [100;999]
5) максимальное значение **s** равно 999 + 999 – 100 – 100 = 1798
6) обратите внимание, что это число не зависит от размера массива

Ответ: 1798.

Пример 19.2 *В программе используется одномерный целочисленный массив A с индексами от 0 до 9. Значения элементов равны 6; 9; 7; 2; 1; 5; 0; 3; 4; 8 соответственно, т.е. A[0] = 6; A[1] = 9 и т.д.*
Определите значение переменной c после выполнения следующего фрагмента программы, записанного ниже на разных языках программирования.
```
c := 0;
for i := 1 to 9 do
  if A[i-1] < A[i] then begin
    c := c + 1;
    t := A[i];
    A[i] := A[i-1];
    A[i-1] := t
  end;
```
Решение:

1) сначала попытаемся понять, что же делает эта программа:
   - в цикле рассматриваются пары соседних элементов, начиная с пары (**A[0]**,**A[1]**) и заканчивая парой (**A[8]**,**A[9]**);
   - если предыдущий элемент **A[i-1]** меньше следующего (**A[i]**), они меняются местами через вспомогательную переменную **t**; таким образом, цикл выполняет один этап (один проход по массиву) метода сортировки массива ***по убыванию***, который называется «методом пузырька»
   - начальное значение переменной **c**, которая нас интересует, равно нулю; при каждой перестановке оно увеличивается на 1 (начиная с нуля), то есть, **c** – счётчик перестановок
2) для первой пары выполняется условие **A[0]< A[1]**, поэтому выполняется перестановка:
   ```
   6 9 7 2 1 5 0 3 4 8
   9 6 7 2 1 5 0 3 4 8
   ```
3) следующая перестановка будет для пары (**A[1]**, **A[2]**):
   ```
   9 6 7 2 1 5 0 3 4 8
   9 7 6 2 1 5 0 3 4 8
   ```
4) следующая – для пары (**A[4]**, **A[5]**):

```
9 7 6 2 1 5 0 3 4 8
9 7 6 2 5 1 0 3 4 8
```

5) и далее еще 3 перестановки, в результате которых значение 0 перемещается до конца массива:

```
9 7 6 2 5 1 3 0 4 8
9 7 6 2 5 1 3 4 0 8
9 7 6 2 5 1 3 4 8 0
```

6) всего было сделано 6 перестановок, при каждой счётчик увеличивался на 1, поэтому после выполнения этого фрагмента значение переменной **c** будет равно 6
7) ответ: 6.

Пример 19.3 *В программе используется одномерный целочисленный массив A с индексами от 1 до 25. Ниже представлен фрагмент программы, в котором задаются значения элементов:*

```
n:= 25;
A[1]:= 2;
for i:= 2 to n do begin
  A[i]:= 2*A[i-1] mod 10;
end;
```

Чему будет равно значение A[25] после выполнения фрагмента программы?

Решение:

1) заметим особенность: внутри цикла берется остаток от деления `2*A[i-1]` на 10, то есть последняя цифра десятичной записи; поэтому все элементы массива – однозначные числа
2) если бы не было этого взятия остатка, каждое последующее число в 2 раза больше предыдущего, цепочка начинается с 2, поэтому в массиве были бы записаны степени числа 2: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 и т.д.
3) выделим последние цифры в этой цепочке:
   2, 4, 8, 6, 2, 4, 8, 6, 2, 4, ...
   они повторяются через 4 элемента
4) таких полных групп в массиве с 25 элементами будет `25 div 4 = 6`; эти 6 групп займут первые 24 элемента, а 25-м будет первый элемент в четвёрке, то есть 2

Ответ: 2.

Пример 19.4 В программе используется одномерный целочисленный массив A с индексами от 0 до 9. Ниже представлен фрагмент программы, в котором значения элементов сначала задаются, а затем меняются.

```
for i:=0 to 9 do
  A[i]:=9-i;
for i:=0 to 4 do begin
  k:=A[i];
  A[i]:=A[9-i];
  A[9-i]:=k;
end;
```

Чему будут равны элементы этого массива после выполнения фрагмента программы?

1) 9 8 7 6 5 4 3 2 1 0
2) 0 1 2 3 4 5 6 7 8 9
3) 9 8 7 6 5 5 6 7 8 9
4) 0 1 2 3 4 4 3 2 1 0

Решение:

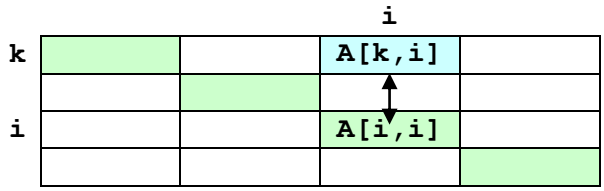
1) выясним, как заполняется массив в первом цикле
```
for i:=0 to 9 do
  A[i]:=9-i;
```
здесь элемент **A[0]** равен 9, элемент **A[1]=8** и т.д. до **A[9]=0**
2) рассмотрим второй цикл, в котором операторы
```
  k:=A[i];
  A[i]:=A[9-i];
  A[9-i]:=k;
```
меняют местами элементы **A[i]** и **A[9-i]**
3) второй цикл выполняется всего 5 раз, то есть останавливается ровно на половине массива
```
for i:=0 to 4 do begin
  ...
end;
```
таким образом в нем меняются элементы A[0]↔A]9], A[1]↔A]8], A[2]↔A]7], A[3]↔A]6] и A[4]↔A]5]
4) в результате массив оказывается «развернут» наоборот, элемент A[0] (он был равен 9) стал последним, следующий (A[1]=8) – предпоследним и т.д., то есть получили
   0 1 2 3 4 5 6 7 8 9
5) Ответ: 2.

Пример 19.5. *Дан фрагмент программы, обрабатывающей двухмерный массив A размера n×n.*
```
k := 1;
for i:=1 to n do begin
  c := A[i,i];
  A[i,i] := A[k,i];
  A[k,i] := c;
end
```
Представим массив в виде квадратной таблицы, в которой для элемента массива A[i,j] величина i является номером строки, а величина j – номером столбца, в котором расположен элемент. Тогда данный алгоритм меняет местами

1) два столбца в таблице
2) две строки в таблице
3) элементы диагонали и k-ой строки таблицы
4) элементы диагонали и k-го столбца таблицы

Решение:

6) сначала разберемся, что происходит внутри цикла; легко проверить (хотя бы ручной прокруткой, если вы сразу не узнали стандартный алгоритм), что операторы
```
  c := A[i,i];
  A[i,i] :=
A[k,i];
  A[k,i] := c;
```
меняют местами значения **A[i,i]** и **A[k,i]**, используя переменную **c** в качестве вспомогательной ячейки;
7) элемент матрицы **A[i,i]**, у которого номера строки и столбца одинаковые, стоит на главной диагонали; элемент **A[k,i]** стоит в том же столбце **i**, но в строке с номером **k**; это значит, что в столбце **i** меняются местами элемент на главной диагонали и элемент в строке **k**



8) так как эти операторы находятся в цикле, где переменная **i** принимает последовательно все значения от 1 до **n**, обмен выполняется для всех столбцов матрицы; то есть, все элементы главной диагонали меняются с соответствующими элементами строки **k**
9) перед циклом стоит оператор присваивания **k := 1;**, а после него переменная **k** не меняется; поэтому в программе элементы главной диагонали обмениваются с первой строкой
10) таким образом, правильный ответ – 3.

Возможные ловушки и проблемы:
- сложность этой задачи в том, что все действия нужно «прокручивать в уме» (или на бумаге), не используя компьютер для отладки
- главная проблема – не перепутать столбцы и строки; номер строки – это (по соглашению) первый индекс элемента матрицы, а номер столбца – второй

Совет:
- чтобы понять, что делает программа, часто бывает полезно сделать ручную прокрутку на матрице небольшого размера, например, 3 на 3 или 4 на 4.
- если матрица небольшая (скажем, 5 на 5) можно (а иногда и нужно) вообще сделать все вычисления вручную и посмотреть, что получится

Пример 19.6. *Значения двух массивов A[1..100] и B[1..100] задаются с помощью следующего фрагмента программы:*

```
for n:=1 to 100 do
  A[n] := (n-80)*(n-80);
for n:=1 to 100 do
  B[101-n] := A[n];
```

Какой элемент массива B будет наибольшим?

1) B[1] 2) B[21] 3) B[80] 4) B[100]

Решение:
1) здесь два цикла, в первом из них заполняется массив **A**, во втором – массив **B**
2) в элемент массива **A[n]** записывается квадрат числа **n-80**; все элементы массива **A** неотрицательны (как квадраты чисел)
3) посмотрим чему равны некоторые элементы массива **A**:
   A[1] = $(1–80)^2$ = $(–79)^2$ = 79^2 A[2] = $(2–80)^2$ = $(–78)^2$ = 78^2
   ...
   A[80] = $(80–80)^2$ = $(0)^2$ = 0 A[81] = $(81–80)^2$ = $(1)^2$ = 1
   ...
   A[99] = $(99–80)^2$ = 19^2 A[100] = $(100–80)^2$ = 20^2
4) таким образом, при увеличении n от 1 до 80 значение A[n] уменьшается от 79^2 до нуля, а потом (для n > 80) возрастает до 20^2
5) отсюда следует, что максимальное значение в массиве A – это A[1] = 79^2
6) во втором цикле для всех номеров n от 1 до 100 выполняется оператор

```
  B[101-n] :=
A[n];
```

который просто переписывает элементы массива A в массив B, «развертывая» массив в обратном порядке (элемент A[1] будет записан в B[100], а A[100] – в B[1])

7) A[1], наибольший элемент массива A, будет записан в B[100], поэтому B[100] – наибольший элемент в массиве B
8) таким образом, правильный ответ – 4.

Пример 19.7. *Значения элементов двухмерного массива A[1..10,1..10] задаются с помощью следующего фрагмента программы:*

```
for i:=1 to 10 do
for k:=1 to 10 do
if i > k then
     A[i,k] := 1
else A[i,k] := 0;
```

Чему равна сумма элементов массива после выполнения этого фрагмента программы?

Решение:

1) в программе есть вложенный цикл, в котором переменная **i** обозначает строку, а **k** – столбец матрицы
2) элементы, для которых **i=k** – это главная диагональ матрицы, поэтому элементы, для которых **i > k** (только они будут равны 1), находятся под главной диагональю
3) в первой строке единичных элементов нет, во второй есть один такой элемент, в третьей – 2, в последней (10-ой) их 9, поэтому сумма элементов массива равна
$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45$$
4) таким образом, правильный ответ – 45.
5) при большом размере массива (например, 100 на 100) суммирование может оказаться трудоемким, поэтому лучше вспомнить формулу для вычисления суммы элементов арифметической прогрессии (именно такая прогрессия у нас, с шагом 1):
$$S = N \cdot \frac{a_1 + a_N}{2},$$
где N - количество элементов, а a_1 и a_N – соответственно первый и последний элементы последовательности; в данном случае имеем
$$S = 9 \cdot \frac{1+9}{2} = 45.$$
6) если приведенная выше формула прочно забыта, можно попытаться сгруппировать слагаемые в пары с равной суммой (как сделал, будучи школьником, великий математик К.Ф. Гаусс), например:
$$1+2+\ldots+9 = (1+9)+(2+8)+\ldots+(4+6)+5 = 4 \cdot 10 + 5$$

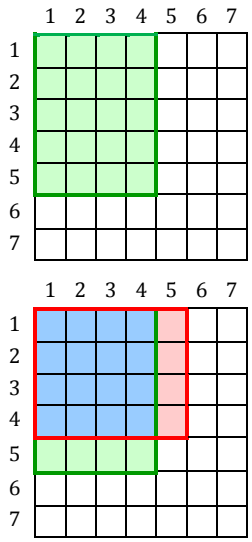
Пример 19.8 *Значения элементов двухмерного массива A[1..10,1..10] сначала равны 5. Затем выполняется следующий фрагмент программы:*

```
for i:=1 to 5 do
  for j:=1 to 4 do begin
    A[i,j]:=A[i,j]+5; { 1 }
    A[j,i]:=A[j,i]+5; { 2 }
```

```
    end;
```

Сколько элементов массива будут равны 10?

Решение (вариант 1, анализ алгоритма):



1) обратим внимание, что в двойном цикле переменная **i** изменяется от 1 до 5, а **j** – от 1 до 4 (на 1 шаг меньше)
2) внутри цикла в операторе, отмеченном цифрой 1 в комментарии, в записи **A[i,j]** переменная **i** – это строка, а **j** – столбец, поэтому по одному разу обрабатываются все элементы массива, выделенные зеленым цветом:
3) это значит, что если оставить только один первый оператор внутри цикла, все выделенные элементы увеличиваются на 5 и станут равны 10
4) теперь рассмотрим второй оператор внутри цикла: в записи **A[j,i]** переменная **i** – это столбец, а **j** – строка, поэтому по одному разу обрабатываются (увеличиваются на 5) все элементы массива, выделенные рамкой красного цвета на рисунке справа
5) теперь хорошо видно, что левый верхний угол массива (квадрат 4 на 4, синяя область) попадает в обе области, то есть, эти 16 элементов будут дважды увеличены на 5: они станут равны 15 после выполнения программы
6) элементы, попавшие в зеленый и красный «хвостики» обрабатываются (увеличиваются на 5) по одному разу, поэтому они-то и будут равны 10
7) всего таких элементов – 8 штук
8) таким образом, правильный ответ – 8.

Решение (вариант 2, прокрутка небольшого массива):

1) понятно, что в программе захватывается только левый верхний угол массива, остальные элементы не меняются
2) сократим размер циклов так, чтобы можно было легко выполнить программу вручную; при этом нужно сохранить важное свойство: внутренний цикл должен содержать на 1 шаг меньше, чем внешний:

```
for i:=1 to 3 do
  for j:=1 to 2 do begin
    A[i,j]:=A[i,j]+5; { 1 }
    A[j,i]:=A[j,i]+5; { 2 }
  end;
```

3) выполняя вручную этот вложенный цикл, получаем

	1	2	3	4	5
1	15	15	10	5	5
2	15	15	10	5	5
3	10	10	5	5	5
4	5	5	5	5	5
5	5	5	5	5	5

4) видим, что в самом углу получился квадрат 2 на 2 (по количеству шагов внутреннего цикла), в котором все элементы равны 15; по сторонам этого квадрата стоят 4 элемента, равные 10, их количество равно удвоенной стороне квадрата
5) в исходном варианте внутренний цикл выполняется 4 раза, поэтому угловой квадрат будет иметь размер 4 на 4; тогда 8 элементов, граничащих с его сторонами, будут равны 10
6) таким образом, правильный ответ – 8.

Возможные проблемы: упрощая задачу, нельзя потерять ее существенные свойства: например, здесь было важно, что внутренний цикл содержит на 1 шаг меньше, чем внешний

И снова массивы в задаче 25 2 части ЕГЭ. Готовясь к заданию 19 разбираемся и с 25 заданием.

Задача 25 (высокий уровень, время – 30 мин)

Тема: Обработка массива (написать программу из 10-15 строк на языке программирования или алгоритм на естественном языке).

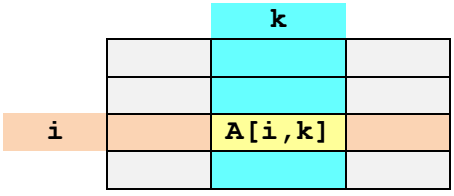
Что нужно знать:

- *массив* – это набор однотипных элементов, имеющих общее имя и расположенных в памяти рядом
- для обращения к элементу массива используют квадратные скобки, запись **A[i]** обозначает элемент массива **A** с номером (индексом) **i**
- для обработки всех элементов массива используется цикл вида[1]

```
for i:=1 to N do begin
  { что-то делаем с элементом A[i] }
end;
```

переменная **i** обозначает номер текущего элемента массива, она меняется от 1 до N с шагом 1, то есть мы "проходим" последовательно все элементы

- *матрица* (двухмерный массив) – это прямоугольная таблица однотипных элементов
- если матрица имеет имя **A**, то обращение **A[i,k]** обозначает элемент, расположенный на пересечении строки **i** и столбца **k**



- каждая строка матрицы – это обычный (одномерный, линейный) массив; для того, чтобы обработать строку **i** в матрице из **M** столбцов, нужно использовать цикл, в котором меняется номер столбца **k**:

```
for k:=1 to M do begin
  { что-то делаем с элементом A[i,k] }
end;
```

- каждый столбец матрицы – это обычный (одномерный, линейный) массив; для того, чтобы обработать столбец **k** в матрице из **N** строк, нужно использовать цикл, в котором изменяется номер строки **i**:

```
for i:=1 to N do begin
  { что-то делаем с элементом A[i,k] }
end;
```

- условие задачи записано на нескольких языках (алгоритмический язык, Паскаль, Бейсик и Си); в принципе, решение можно писать и на любом другом языке, в том числе на естественном языке или в виде блок-схемы.

При решении задачи 25 ЗАПРЕЩАЕТСЯ использовать библиотечные подпро-

[1] По традиции нумерация элементов массива в Паскале обычно начинается с единицы, далее N обозначает размер массива (количество элементов).

граммы языка программирования, например, подпрограммы сортировки и т.п.

Задача 25.1. *Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от –10 000 до 10 000 включительно. Опишите на естественном языке или на одном из языков программирования алгоритм, позволяющий найти и вывести количество пар элементов массива, сумма которых нечётна и положительна. Под парой подразумевается два подряд идущих элемента массива. Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но использовать все описанные переменные не обязательно.*

Паскаль	Алгоритмический язык

`const N = 20;`
`var a: array [1..N] of integer;`
`  i, j, k: integer;`
`begin`
`  for i := 1 to N do`
`  readln(a[i]);`
`  ...`
`end.`
	`алг`
	`нач`
	`  цел N = 20`
	`  цел таб a[1:N]`
	`  цел i, j, k`
	`  нц для i от 1 до N`
	`  ввод a[i]`
	`  кц`
	`  ...`
	`кон`

Решение:

7) даже если вы хорошо владеете программированием, сначала лучше (прежде всего, для себя) написать алгоритм на русском языке
8) в задании нужно перебрать все пары соседних элементов, начиная от пары **(a[1], a[2])** до пары **(a[N-1],a[N])** и подсчитать, для скольких пар сумма элементов нечётна и положительна
9) поскольку нам нужно что-то считать, придётся использовать переменную счётчик**;** сначала её значение равно 0, затем, при каждой найденной подходящей паре, значение счётчика увеличивается на 1**;** в качестве счётчика можно использовать любую целую переменную из тех, что были объявлены в условии, например, **k**:
```
k := 0;
перебрать все пары
  если пара подходящая, то
    k := k + 1
```
10) как же перебрать все пары? примем, что переменная **i** будет хранить номер первого элемента в паре, то есть, будем рассматривать пары **(a[i],a[i+1])**
11) очевидно, что нужно организовать цикл, который перебирает все значения **i** в интервале от 1 (для первой пары, **(a[1], a[2])**) до N-1 (для последней пары, **(a[N-1],a[N])**)
```
for i:=1 to N-1 do ...
```
12) как определить, что пара **(a[i],a[i+1])** подходящая? нужно вычислить её сумму **a[i]+a[i+1]** и проверить, верно ли, что эта сумма нечётна и положительна
13) нечётность проверим, вычислив остаток от деления на 2**;** если этот остаток не равен 0, число нечётное
14) получается такой цикл, после которого нужно не забыть вывести результат – значение счётчика **k**:
```
k := 0;
for i:=1 to N-1 do
```

```
  if (a[i]+a[i+1] > 0) and ((a[i]+a[i+1]) mod 2 <> 0)
then
    k := k + 1;
writeln(k);
```

15) для того, чтобы не вычислять дважды сумму в условном операторе, можно было записать её в свободную переменную **j**:

```
k := 0;
for i:=1 to N-1 do begin
  j := a[i]+a[i+1];
  if (j > 0) and (j mod 2 <> 0) then
    k := k + 1
end;
writeln(k);
```

16) возможен и ответ на естественном языке (русском):

> Записать значение 0 в переменную k. Затем в цикле перебрать все пары соседних элементов, начиная с пары (a[1],a[2]) до пары (a[N-1],a[N]). Для каждой пары вычислить сумму, если эта сумма положительна и нечётна (остаток от её деления на 2 не равен нулю), увеличить значение k на единицу. После завершения цикла вывести значение k.

Возможные проблемы:

- не забудьте сказать, что нужно вывести после окончания работы программы
- не забудьте задать правильное начальное значение для счётчика
- не забудьте, что в Паскале каждое из простых условий (отношений) нужно брать в скобки
- не забудьте, что если в теле цикла нужно выполнить несколько операторов, их нужно заключать в "операторные скобки" **begin-end**
- цикл по **i** должен заканчиваться не на N, а на N-1, чтобы не было выхода за границы массива (рассматриваются элементы **a[i]** и **a[i+1]**)
- если вы достаточно хорошо владеете русским языком для того, чтобы понятно излагать свои мысли, с точки зрения тактики рекомендуется писать алгоритм на русском языке – по крайней мере, тут не снизят балл за пропущенную точку с запятой
- просмотрите внимательно диапазон, в котором находятся исходные числа; дело в том, что во многих языках, например, в Паскале и в Си, остаток от деления отрицательного числа на положительное – число отрицательное, например **(-7) mod 2 = -1**, поэтому определять, например, "неделимость" элемента массива на 2 с помощью условия **a[i] mod 2 = 1** нельзя (не будет работать для отрицательных чисел), нужно использовать условие **a[i] mod 2 <> 0.**

Задача 25.2. *Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от 0 до 1000. Опишите на русском языке или на одном из языков программирования алгоритм, позволяющий найти и вывести минимальное значение среди элементов массива, которые имеют чётное значение и не делятся на три. Гарантируется, что в исходном массиве есть хотя бы один элемент, значение которого чётно и не кратно трем.*
Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но использовать все описанные переменные не обязательно.

Естественный язык:

Объявляем массив A из 20 элементов.
Объявляем целочисленные переменные I, J, MIN.
В цикле от 1 до 20 вводим элементы массива A с 1-го по 20-й.

Паскаль:

```
const N=20;
var a: array [1..N] of integer;
    i, j, min: integer;
begin
  for i:=1 to N do
    readln(a[i]);
   …
end.
```

Решение:

1) даже если вы хорошо владеете программированием, сначала лучше (прежде всего, для себя) написать алгоритм на русском языке
2) здесь требуется найти минимальный элемент из всех, которые имеют чётное значение и не делятся на 3
3) делимость одного целого числа на другое проверяется с помощью операции взятия остатка (в Паскале – операция **mod**): первое число делится на второе, если остаток от деления равен 0
4) тогда условие, определяющее отбор нужных элементов, запишется в виде
   ```
   (a[i] mod 2 = 0) and (a[i] mod 3 <> 0)
   ```
5) стандартный цикл поиска минимального элемента, удовлетворяющего условию, выглядит так:
   ```
   for i:=1 to N do
     if <условие верно> and (a[i] < min) then
       min := a[i];
   ```
6) остается один вопрос: каким должно быть начальное значение переменной **min**? его нужно выбрать таким, чтобы для первого же "подходящего" элемента выполнилось условие **a[i] < min**, и это "временное" начальное значение было бы заменено на реальное
7) к счастью, диапазон входных чисел ограничен (от 0 до 1000), поэтому можно выбрать любое значение, больше 1000, например, 1001 или 9999[2]
8) таким образом, решение задачи на естественном языке выглядит так:
   Записываем в переменную **min** *значение 1001.*
   Затем в цикле просматриваем все элементы массива, с первого до последнего. Если остаток от деления очередного элемента на 2 равен 0 и остаток от его деления на 3 не равен нулю и значение элемента меньше, чем значение переменной **min**, *записать в переменную* **min** *значение рассматриваемого элемента массива. Затем переходим к следующему элементу.*
   После окончания работы цикла выводим значение переменной **min**.
9) аналогичное решение на Паскале:
   ```
   min:=1001;
   for i:=1 to N do
   if (a[i] mod 2=0) and (a[i] mod 3 <> 0) and (a[i]<min) then
   ```

[2] Вообще говоря, в данной задаче не требуется находить номер минимального элемента, поэтому сначала можно записать в переменную **min** число 1000 – проверьте, что программа все равно выдаст верное значение.

```
        min:=a[i];
    writeln(min);
```

Возможные проблемы:

- не забудьте сказать, что нужно вывести после окончания работы программы

если вы достаточно хорошо владеете русским языком для того, чтобы понятно излагать свои мысли, с точки зрения тактики рекомендуется писать алгоритм на русском языке – по крайней мере, тут не снизят за пропущенную точку с запятой

просмотрите внимательно диапазон, в котором находятся исходные числа; дело в том, что во многих языках, например, в Паскале и в Си, остаток от деления отрицательного числа на положительное – число отрицательное, например (-7) mod 3 = -1, поэтому определять, например, ″неделимость″ элемента массива на 3 с помощью условия a[i] mod 3 = 1 нельзя (не будет работать для отрицательных чисел), нужно использовать условие a[i] mod 3 <> 0.

Задача 25.3*Опишите на русском языке или одном из языков программирования алгоритм получения из заданного целочисленного массива размером 30 элементов другого массива, который будет содержать модули значений элементов первого массива (не используя специальной функции, вычисляющей модуль числа). Полученный массив нужно вывести на экран.*

Решение:

1) даже если вы хорошо владеете программированием, сначала лучше (прежде всего, для себя) написать алгоритм на русском языке (или на псевдокоде – это нечто среднее между словесным алгоритмом и готовой программой)
2) по условию нужно выделить в памяти два массива одинакового размера, назовем их **A** и **B;** обозначим размер массивов через **N**, индексы элементов изменяются от 1 до **N;**
3) в цикле в каждый элемент **B[i]** массива **B** нужно записать модуль соответствующего элемента **A[i]** массива **A**, это нужно сделать для всех **i** от 1 до **N**
4) есть небольшая сложность: запрещено использовать стандартную функцию вычисления модуля**;** согласно определению модуля решение может быть такое: если элемент **A[i]** больше или равен нулю, записываем в **B[i]** его значение без изменений, а если меньше нуля – меняем знак, то есть, в **B[i]** записываем (**-A[i]**)
5) решение в виде алгоритма на русском языке может выглядеть так: **″**Выделяем в памяти второй массив того же размера. В цикле рассматриваем все элементы первого массива с первого до последнего. Если текущий элемент больше нуля или равен нулю, в соответствующий элемент второго массива записываем его значение без изменений**;** если текущий элемент меньше нуля, во второй массив записываем значение элемента с обратным знаком. Выводим второй массив на экран**″**.
6) осталось написать программу, практически дословно реализующую это решение:

```
    const N = 30;
    var a, b:array[1..N] of integer;
        i: integer;
    begin
    for i:=1 to N do  { ввод всех элементов массива с клавиатуры }
      read(a[i]);
```

```
for i:=1 to N do  { формирование массива B }
  if a[i]<0 then
       b[i]:=-a[i]
  else b[i]:=a[i];
writeln('Результат:');
for i:=1 to N do  { вывод всех элементов массива B }
  write(b[i], ' ');
end.
```

7) размер массива грамотно задавать через константу (`const N=30;`), а не вписывать число в каждый цикл**;** тогда, если нужно будет переделать программу для массива другого размера, достаточно будет изменить всего одно число в начале программы

Возможные проблемы:
- проверяйте правильность минимального и максимального значения переменной цикла в заголовке цикла **for**
- не забывайте вывести результат в конце работы программы

Задача 25.4. *Опишите на русском языке или одном из языков программирования алгоритм подсчета максимального количества подряд идущих совпадающих элементов в целочисленном массиве длины 30.*

Решение:

1) сначала нужно понять задачу**;** предположим, что в массиве есть одинаковые элементы, стоящие рядом:

1	1	2	2	1	1	1	1	3	3	2	2

2) самая длинная цепочка стоящих рядом элементов в данном случае состоит из 4-х единиц (она выделена желтым фоном)
3) нам нужно по крайней мере две переменных: для хранения номера текущего элемента (при обработке массива в цикле) и для хранения максимального количества идущих подряд элементов (обозначим ее **kMax**)
4) в целом (пока неточный) алгоритм может выглядеть так: **"**пройти весь массив, подсчитывая для каждого элемента длину цепочки подряд идущих одинаковых чисел, если эта длина больше **kMax**, то записать ее в **kMax"**
5) отсюда сразу следует, что необходима еще одна переменная (обозначим ее через **k**), показывающая для каждого элемента массива длину цепочки одинаковых чисел, которая заканчивается на этом элементе:

	1	1	2	2	1	1	1	1	3	3	2	2
k	1	2	1	2	1	2	3	4	1	2	1	2
kMax	1	2	2	2	2	2	3	4	4	4	4	4

6) следующий шаг к решению: нужно понять, как изменять переменную **k** при проходе по массиву**;** можно сделать так: если очередной элемент равен предыдущему, счетчик **k** увеличиваем на единицу, а если не равен – записываем в него 1 (цепочка одинаковых чисел кончилась, началась новая, в ней пока один элемент)
7) при таком подходе проблема может возникнуть при просмотре первого элемента, потому что для него нет предыдущего**;** поэтому описанную выше процедуру будем в цикле применять ко всем элементам массива, начиная **со второго** (а не с первого)**;** в самом начале программы запишем в **k** и **kMax** по единице –

таким образом, мы "вручную" (без цикла) рассмотрели первый элемент массива

8) теперь можно написать алгоритм на русском языке:
"Выделим две вспомогательные переменные, **k** и **kMax**, и запишем в каждую из них по единице. В цикле рассматриваем все элементы массива со **второго** до последнего, если очередной элемент равен предыдущему, увеличиваем **k;** если **k > kMax**, записываем в **kMax** значение **k**. В конце цикла в **kMax** окажется требуемое значение".
9) этот алгоритм реализуется в такой программе:

```
const N =30;
var a: array[1..N] of integer;
  i, k, kMax: integer;
begin
for i:=1 to N do readln(A[i]); { ввод массива }
k := 1;                        { обрабатываем A[1] }
kMax := 1;
for i:=2 to N do begin       { а теперь в цикле A[2]...A[N} }
  if A[i] = A[i-1] then        { цепочка продолжается }
       k := k + 1
  else k := 1;                 { цепочка закончилась }
  if k > kMax then kMax := k;
end;
writeln(kMax);
end.
```

Возможные проблемы:

- как видим, основная сложность в этой задаче – не написать программу, а придумать хороший (часто еще нужно – быстрый) алгоритм
- проверьте, что будет записано в переменные до начала цикла (определены ли их начальные значения)
- проверяйте, не выйдет ли индекс за границу массива в начале или в конце цикла
- будьте внимательны с "крайними" случаями, например, нужно обязательно убедиться, что программа работает, когда интересующая нас цепочка стоит в самом начале или в самом конце массива

Задача 25.5*Дан целочисленный квадратный массив 10 x 10. Опишите на русском языке или на одном из языков программирования алгоритм вычисления суммы максимальных элементов из каждой строки. Напечатать значение этой суммы. Предполагается, что в каждой строке такой элемент единственный.*

Решение:

1) суть задачи: среди элементов каждой строки нужно выбрать максимальный, и все эти выбранные значения сложить
2) несложно сразу написать алгоритм на русском языке:
"Чтобы накапливать сумму, нужно ввести целую переменную **Sum**, в которую в самом начале записываем 0 **;** далее в цикле просматриваем все строки, для каждой строки находим максимальный элемент и прибавляем его значение к **Sum**. Для определения максимального элемента в строке вводим переменную **max** и сначала записываем в нее значение первого элемента этой строки. Затем в цик-

ле просматриваем все остальные элементы, начиная со **второго** до конца массива. Если очередной элемент больше значения **max**, записываем в **max** значение этого элемента".

3) сначала напишем программу на псевдокоде:

```
const N=10;
{ ввод матрицы N на N }
Sum := 0;
for i:=1 to N do begin
  { max := максимальный элемент в i-ой строке }
  Sum := Sum + max;
end;
```

4) остается записать на Паскале те части, которые взяты в фигурные скобки, и до конца оформить программу; по правилам ЕГЭ можно не писать в программе команды для ввода массива, поэтому мы оставим на этом месте комментарий:

```
const N=10;
var A: array[1..N,1..N] of integer;
    i, k, max, Sum: integer;
begin
{ ввод матрицы N на N }
Sum := 0;
for i:=1 to N do begin
  max := A[i,1];
  for k:=2 to N do
    if A[i,k] > max then max := A[i,k];
  Sum := Sum + max;
end;
writeln(Sum);
end.
```

Возможные проблемы:

1. проверьте, правильно ли заданы (и заданы ли вообще) начальные значения для всех переменных
2. проверьте, правильно ли расставлены операторные скобки begin-end, ограничивающие тело цикла; их обязательно нужно ставить, если в теле цикла несколько операторов
3. проверяйте, не выйдет ли индекс за границу массива в начале или в конце цикла
4. не перепутайте номер строки (это первый индекс) и номер столбца (второй индекс)
5. для надежности не рекомендуется использовать в одной программе переменные i и j, потому что они слишком похоже выглядят,

вот пример **ошибочного решения** в этой задаче:

```
for i:=1 to N do begin
  max := A[i,1];
  for i:=2 to N do
    if A[j,i] > max then max := A[i,j];
  Sum := Sum + max;
end;
```

если вы все же используете переменные i и j, нужно писать их очень четко, чтобы они отличались друг от друга

Немного тактики:

- в этом задании можно написать алгоритм на русском языке, а можно (вместо этого)

написать компьютерную программу на одном из языков программирования

- если вы хорошо умеете выражать свои мысли по-русски, ~~собаководы~~ эксперты рекомендуют писать **только** алгоритм на русском языке**;** дело в том, что если вы сделаете много ошибок в программе, оценка будет снижена даже при абсолютно правильном алгоритме
- если вам сложно изъясняться на родном языке, а легче записать свои мысли на Паскале или Си – пишите программу, но тщательно проверяйте ее на предмет возможных случайных ошибок-опечаток, которые можно сделать просто по невнимательности:
  - задавайте все начальные значения для переменных
  - проверяйте правильность написания ключевых слов
  - если в теле цикла несколько операторов, заключайте их в блок **`begin-end`** (операторные скобки)
  - проверяйте начальное и конечное значение переменной цикла
  - если вы используете циклы while или repeat, проверьте, что переменная цикла изменяется в теле цикла (иначе в программе будет зацикливание, а на ЕГЭ – потерянные баллы)
  - выводите на экран именно то, что требуется по условию
  - ставьте точку с запятой в конце операторов
  - НЕ ставьте точку с запятой перед **`else`** (Паскаль)
  - ставьте точку в конце последнего оператора **`end`** (Паскаль)

За что снимают баллы:

- задано неверное начальное значение переменных (или вообще не задано)
- неверно указано условие завершения цикла
- **"**забыли**"** изменять переменную цикла в цикле **`while`** (**`repeat`**)
- перепутаны знаки < и >, логические операции **`or`** и **`and`**
- неверно расставлены операторные скобки **`begin-end`**
- программа не выводит результат или выводит не то, что спрашивают
- синтаксические ошибки (знаки пунктуации – запятые, точки, точки с запятой**;** неверное написание ключевых слов) допускаются в разумных пределах (если они не искажают замысел автора)

Задание 21 (повышенный уровень, время – 6 мин)

Тема: Анализ программы с подпрограммами.

Что нужно знать:

- функция – это вспомогательный алгоритм, который возвращает некоторое значение–результат
- в Паскале функция располагается выше основной программы и оформляется следующим образом (вместо многоточия могут быть любые операторы):

```
function F(x: integer):integer;
begin
  ...
  F:= <результат функции>
end;
```

- в заголовке функции записывают имя функции, в скобках – список параметров, далее через двоеточие – тип возвращаемого значения; в приведенном примере функция **F** принимает один целый параметр, к которому внутри функции нужно обращаться по имени **x**, и возвращает целое число
- результат функции записывается в специальную переменную, имя которой совпадает с именем функции; объявлять эту переменную не нужно
- если параметров несколько, для каждого из них указывают тип:

```
function F(x: integer; y: integer):integer;
```

- если несколько соседних параметров имеют одинаковый тип, можно их объединить в список:

```
function F(x, y: integer):integer;
```

- следующая программа ищет наименьшее значение функции **F(x)** на интервале **[a,b]**, просматривая значения от **a** до **b** с шагом 1:

```
M:=a; R:=F(a);
for t:=a to b do
  if F(t) < R then begin
    R:=F(t); M:=t;
  end;
```

- цикл для поиска наибольшего значения выглядит точно так же, только знак < нужно заменить на знак >
- если функция представляет собой квадратный трехчлен вида $F(x) = ax^2 + bx + c$, то абсцисса, соответствующая точке минимума, вычисляется по формуле

$$x_{\min} = \frac{-b}{2a}$$

этот результат можно получить (вывести, если забыли), например, так:

  - в критической точке (точке минимума, точке максимума или точке перегиба) производная функции обращается в 0;
  - находим производную $F'(x) = 2ax + b$
  - приравниваем ее к нулю: $2ax + b = 0 \quad \Rightarrow \quad x = -\frac{b}{2a}$.

- если квадратный трехчлен задан в виде $F(x) = a(x - p)(x - q)$, то абсцисса, соответствующая точке минимума, вычисляется по формуле

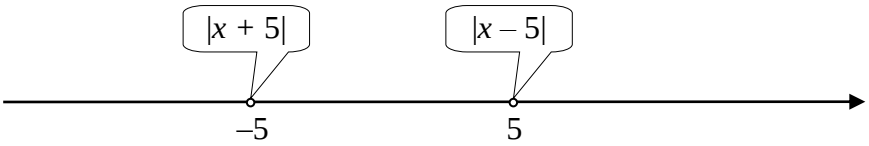
$$x_{\min} = \frac{p+q}{2}$$

Задача 21.1 *Определите, какое число будет напечатано в результате выполнения следующего алгоритма:*

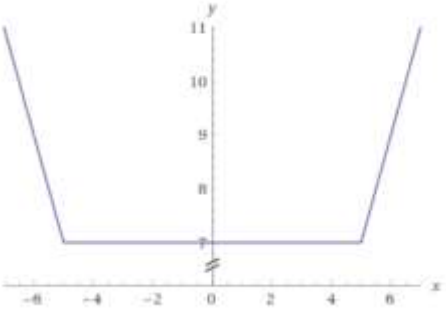
```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:= abs( abs(x-5) + abs(x+5) - 3) + 12;
end;
BEGIN
  a:=-20; b:=20;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)<R)then begin
      M:=t;
      R:=F(t);
    end;
  end;
  write(M+R);
END.
```

Решение:

1) заметим, что в программе есть цикл, в котором переменная **t** принимает последовательно все целые значения в интервале от **a** до **b**:
```
for t:=a to b do begin
  ...
end;
```
2) до начала цикла в переменную M записывается значение **a**, а в переменную **R** – значение функции в точке **a**:
```
M:=a; R:=F(a);
```
3) внутри цикла есть условный оператор, в котором вычисляется значение функции **F(t)** и сравнивается со значением переменной **R**:
```
if (F(t)<R)then begin
  M:=t;
  R:=F(t);
end;
```
если новое значение функции меньше, чем значение **R**, в **R** записывается значение функции в точке **t**, а в переменной **M** запоминается само значение t (аргумент функции, соответствующий значению в **R**)
4) в результате анализа пп. 1-3 можно сделать вывод, что цикл ищет минимум функции **F(t)** на интервале от **a** до **b**, и после выполнения цикла в переменной M оказывается значение аргумента t, при котором функция достигает минимума на заданном интервале (здесь это интервал [-20, 20])
5) запишем заданную функцию в привычном «математическом» виде:
$f(x) = | | x - 5 | + | x + 5 | - 3 | + 12$
чтобы найти минимум этой функции без ручного перебора всех целых значений x, нам нужно представлять, как идёт её график
6) сначала рассмотрим функцию под знаком «большого» модуля $g(x) = | x - 5 | + | x + 5 | - 3$;
7) находим нули выражений под знаком модуля и отмечаем их на числовой оси:



8) раскрываем модули отдельно для каждого интервала; рассматриваем интервал (–∞; –5), раскрываем модули (оба с обратным знаком):
   $g(x) = -(x-5) - (x+5) - 3 = -2x - 3$
   на этом интервале функция убывает
9) рассматриваем полуинтервал [–5; 5), раскрываем модули:
   $g(x) = -(x-5) + (x+5) - 3 = 7$
   это значит, что на полуинтервале [–5; 5) функция принимает постоянное значение
10) рассматриваем полуинтервал [5; ∞), раскрываем модули:
    $g(x) = (x-5) + (x+5) - 3 = 2x - 3$
    на этом интервале функция возрастает
    график функции показан на рисунке справа



11) как мы показали, на всей числовой оси функция $g(x)$ принимает положительные значения, так что $|g(x)| = g(x)$, поэтому $f(x) = |g(x)| + 12 = g(x) + 12$
12) во всех точках на отрезке [-5; 5] функции $f(x)$ равно 7 + 12 = 19, поэтому программа найдёт одну из этих точек как точку минимума
13) осталось понять, какую именно точку она найдёт; посмотрим на программу: запоминание новой точки минимума происходит только тогда, когда только что вычисленное значение F(t) станет **строго меньше**, чем хранящееся в переменной R:

```
if (F(t)< R)then begin
  M:=t;
  R:=F(t);
end;
```

14) поэтому после нахождения точки минимума $x = -5$ никаких изменений не произойдет, и в переменной M останется значение «–5»; таким образом, будет найдет первый минимум
    (*Примечание*: если бы в условии было нестрогое неравенство (<=), была бы найдена последняя из точек с минимальной ординатой, M = 5)
15) обратим внимание, что на экран выводится не M, а M+R, поэтому результат будет равен
    (–5) + 19 = 14
16) Ответ: 14.

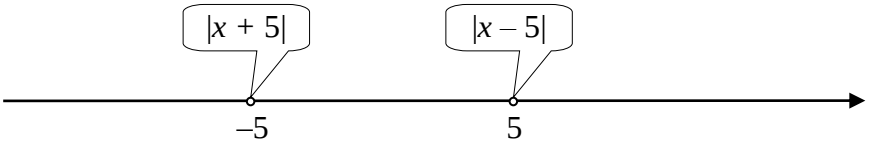
Задача 21.2 *Определите, какое число будет напечатано в результате выполнения следующего алгоритма:*

```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:= abs( abs(x-5) + abs(x+5)- 20 ) + 4;
end;
BEGIN
  a:=-20; b:=20;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)<R)then begin
```

```
        M:=t;
        R:=F(t);
      end;
    end;
    write(M+R);
END.
```

Решение:

1. заметим, что в программе есть цикл, в котором переменная **t** принимает последовательно все целые значения в интервале от **a** до **b**:
   ```
   for t:=a to b do begin
     ...
   end;
   ```
2. до начала цикла в переменную M записывается значение **a**, а в переменную **R** – значение функции в точке **a**:
   ```
   M:=a; R:=F(a);
   ```
3. внутри цикла есть условный оператор, в котором вычисляется значение функции **F(t)** и сравнивается со значением переменной **R**:
   ```
   if (F(t)<R)then begin
     M:=t;
     R:=F(t);
   end;
   ```
   если новое значение функции меньше, чем значение **R**, в **R** записывается значение функции в точке **t**, а в переменной **M** запоминается само значение t (аргумент функции, соответствующий значению в **R**)
4. в результате анализа пп. 1-3 можно сделать вывод, что цикл ищет минимум функции **F(t)** на интервале от **a** до **b**, и после выполнения цикла в переменной M оказывается значение аргумента t, при котором функция достигает минимума на заданном интервале (здесь это интервал [-20, 20])
5. запишем заданную функцию в привычном «математическом» виде:
   $f(x) = | | x - 5 | + | x + 5 | - 20 | + 4$
   чтобы найти минимум этой функции без ручного перебора всех целых значений x, нам нужно представлять, как идёт её график
6. сначала рассмотрим функцию под знаком «большого» модуля $g(x) = | x - 5 | + | x + 5 | - 20$;
7. находим нули выражений под знаком модуля и отмечаем их на числовой оси:



8. рассматриваем интервал $(-\infty; -5)$, раскрываем модули (оба с обратным знаком):
   $g(x) = -(x - 5) - (x + 5) - 20 = -2x - 20$
   на этом интервале функция $g(x)$ убывает
9. рассматриваем полуинтервал $[-5; 5)$, раскрываем модули:
   $g(x) = -(x - 5) + (x + 5) - 20 = -10$
   на этом интервале значение функции $g(x)$ постоянно
10. рассматриваем полуинтервал $[5; \infty)$, раскрываем модули:
    $g(x) = (x - 5) + (x + 5) - 20 = 2x - 20$
    на этом интервале функция $g(x)$ возрастает

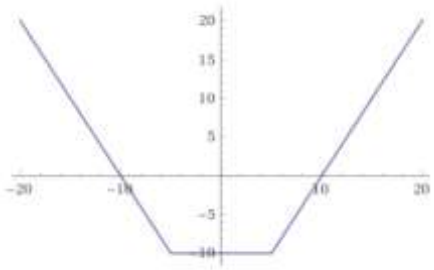
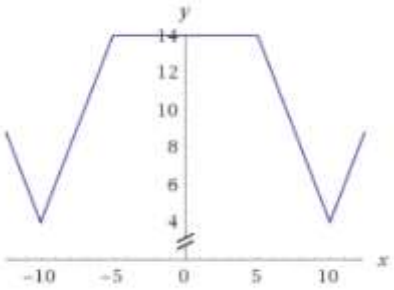


график функции $g(x)$ показан на рисунке справа



11. как видно по графику, на интервале (–10; 10) функция имеет отрицательные значения; найти границы этого интервала можно решив уравнения
    $g(x) = -2x - 20 = 0 \Rightarrow x = -10$
    $g(x) = 2x - 20 = 0 \Rightarrow x = 10$
12. поскольку $f(x) = |\ g(x)\ | + 4$, за счёт модуля «язык», вылезший вниз за ось OX, загнётся вверх, так что функция $f(x)$ будет иметь минимальные значения, равные 4, как раз в точках $x = -10$ и $x = 10$ (см. рисунок справа)
13. осталось понять, какую именно точку она найдёт; посмотрим на программу: запоминание новой точки минимума происходит только тогда, когда только что вычисленное значение F(t) станет **строго меньше**, чем хранящееся в переменной R:
```
if (F(t) < R)then begin
  M:=t;
  R:=F(t);
end;
```
14. поэтому в точке второго минимума $x = 10$ никаких изменений не произойдет, и в переменной M останется значение «–10»; таким образом, будет найдет первый минимум (*Примечание*: если бы в условии было нестрогое неравенство (<=), была бы найдена последняя из точек с минимальной ординатой, M = 10)
15. обратим внимание, что на экран выводится не M, а M+R, поэтому результат будет равен
    (–10) + 4 = – 6
16. Ответ: – 6.

Задача 21.3. *Напишите в ответе число, равное количеству различных значений входной переменной k, при которых приведённая ниже программа выводит тот же ответ, что и при входном значении k = 25. Значение k = 25 также включается в подсчёт количества различных значений k.*

```
var k, i : longint;
function f(n: longint) : longint;
begin
  f := n * n * n;
end;
begin
  readln(k);
  i := 1;
  while f(i) < k do
    i := i+1;
  if f(i)-k <= k-f(i-1) then
       writeln(i)
  else writeln(i-1)
end.
```

Определяем, что будет напечатано при $k = 25$.

i	1	2	3	
$f(i)$	1	8	27	
$f(i) < k$	+	+	–	
$f(i) - k \le k - f(i\text{-}1)$				27-25 ≤ 25-8, верно
Вывод на экран				3

1) Следовательно, последнее значение переменной i может быть равно 3, если условие $f(i) - k \le k - f(i-1)$ истинно или может быть равно 4, если условие ложно. Т.е. при i=3 должно выполняться условие $f(i) - k \le k - f(i-1)$, а при $i = 4$ должно выполняться условие $f(i) - k > k - f(i-1)$.
2) Найдем левую границу. Преобразуем первое выражение:
   $f(3) - k \le k - f(3-1)$
   $f(3) + f(2) \le 2k$
   $27 + 8 \le 2k$
   $k \ge \frac{35}{2}$
   Поскольку значение k должно быть целым, то $k \ge 18$.
3) Найдем правую границу. Преобразуем второе выражение:
   $f(4) - k > k - f(4-1)$
   $f(4) + f(3) > 2k$
   $64 + 27 > 2k$
   $k < \frac{91}{2}$
   Поскольку значение k должно быть целым, то $k \le 45$.
4) Таким образом, $k \in [18,45]$, т.е всего 28 различный значений.
   Ответ: 28.

Задача 21.4.. *Напишите в ответе количество различных значений входной переменной a из интервала от 1 до 100 (включая границы), при которых программа выдаёт тот же ответ, что и при входном значении a = 20. Значение a = 20 также включается в подсчёт различных значений a.*

```
var i, k,a: integer;
function f(x: integer): integer;
begin
  if x >1 then
    f := x mod 2 * f(x div 2)
  else
    f := x;
end;
begin
  k := 0;
  readln(a);
  for i := 1 to a do
      if f(i) =1 then k:=k+1;
  writeln(k);
end.
```

Решение:

1) Рассмотрим, как работает функция, приведенная в программе. Заметим, что (**x mod 2**) – младшая цифра двоичного представления числа **x** в двоичной системе счисления, (**x div 2**) – число **x** без младшей своей цифры в двоичном представлении.
2) Таким образом, функция находит произведение цифр числа в двоичном представлении. Программа в целом находит количество чисел, произведение цифр в двоичной записи которых равно 1, то есть таких чисел, двоичная запись которых не содержит нулей.

3) В общем виде такие числа можно представить, как 2^n-1, где **n** – натуральное число. В диапазоне от 1 до 20 таких чисел 4: 1, 3, 7, 15. А в диапазоне от 1 до 100 – 6: 1, 3, 7, 15, 31, 63.
4) Таким образом, искомые числа – это числа от 15 до 30.
5) Ответ: 16.

Задача 21.5. *Какое число будет напечатано в результате выполнения программы?*

```
var i, k: integer;
function f(x: integer): integer;
begin
  if x > 0 then
    f := x mod 2 + f(x div 2)
  else
    f := 0;
end;
begin
  k := 0;
  for i := 0 to 1023 do
    if f(i mod 32) = 1 then
      if f(i div 32) = f(i mod 32) then k := k + 1;
  writeln(k);
end.
```

Решение:

1) Рассмотрим, как работает функция, приведенная в программе. Заметим, что (**x mod 2**) – младшая цифра двоичного представления числа **x** в двоичной системе счисления, (**x div 2**) – число x без младшей своей цифры в двоичном представлении, (**i mod 32**) - число, составленное из пяти младших битов двоичной записи числа **i**, (**i div 32**) – число, составленное из пяти старших битов двоичной записи числа **i**. Таким образом, функция считает сумму цифр двоичной записи аргумента.
2) В основной части программы рассматриваются числа от 1 до 1023, счетчик увеличивается на 1, если сумма цифр в младших пяти разрядах двоичного представления числа равна 1, и сумма цифр в старших пяти разрядах двоичного представления числа тоже равна 1.
3) Таким образом, необходимо найти количество чисел, в двоичной записи которых в младших пяти разрядах и в старших пяти разрядах находится ровно по одной единице.
4) Дополняя двоичное представление чисел до 10 битов, получим, что старшая единица может располагаться на 5 позициях; в каждом таком случае младшая единица может тоже располагаться на пяти позициях. Тогда количество таких чисел равно 5*5=25.
5) Ответ: 25.

Задача 21.6. *Сколько существует таких четырехзначных натуральных чисел, что при вводе их программа выведет такое же число, что и при вводе числа 1133. Значение 1133 также включается в подсчёт.*

```
function f(x: integer): integer;
begin
  if x mod 2 =1 then
    f := x mod 10 + f(x div 10)
  else
```

```
    f := 0;
end;
var N: integer;
begin
  readln(n);
  writeln(f(N));
end.
```

Решение:

1) В первую очередь необходимо разобраться, как работает описанная в программе функция. Заметим, что функция рекурсивная, (**x div 10**) – это число **x** без последней своей цифры, а (**x mod 2**) – остаток от деления **x** на 2, то есть признак делимости этого числа (и его младшей цифры) на 2.
2) Если на вход функции подается число, младшая цифра которого четная, то функция возвращает 0, иначе она возвращает сумму этой цифры и значения функции от аргумента без последней цифры. Если, например, подать на вход функции число, содержащее только нечетные цифры, она возвратит сумму эти цифр (при целочисленном делении на 10 старшей цифры получится 0, что завершит рекурсию). Таким образом, функция **f** возвращает сумму первой непрерывной последовательности нечетных цифр числа **x**, начиная с младшей.
3) Для входного числа 1133 значение функции равно 8. Это значение можно было получить и при ручном выполнении программы, но это не дало бы понимания работы функции.
4) Найдем в требуемом диапазоне числа, у которых сумма непрерывной последовательности нечетных цифр, начиная с младшей, равна 8. Сумма состоит не более, чем из четырех слагаемых. Покажем, что при разложении числа 8 на нечетные слагаемые могут возникнуть суммы только из двух или четырех цифр: действительно, если сложить три нечетных числа, то и сумма будет нечетной, а единственное нечетное число не может быть равным 8.
5) Сначала найдем разложение числа 8 на сумму четырех нечетных цифр: 8=1+1+3+3. Количество чисел, состоящих из этих цифр, равно 6. Другое разложение: 8=5+1+1+1; количество чисел, составленных из этих цифр, равно 4.
6) Теперь найдем разложения числа 8 на два нечетных слагаемых: 8=5+3 и 8=7+1. Старшая цифра в искомых числах может принимать 9 значений (от 1 до 9), вторая цифра – любое четное значение (5 штук). Таким образом, количество чисел с последовательностью нечетных цифр, начиная с младшей, длиной 2 и суммой 8 есть 9*5*2*2=180.
7) Ответ: 190.

Задача 21.7. Напишите в ответе наименьшее значение входной переменной k, при котором программа выдаёт тот же ответ, что и при входном значении k = 10.

```
var k, i : longint;
function f(n: longint): longint;
begin
  f := n * n * n;
end;
function g(n: longint): longint;
begin
  g := 2*n + 3;
end;
```

```
begin
  readln(k);
  i := 1;
  while f(i) < g(k) do
    i := i+1;
  writeln(i)
end.
```

Решение:

1) сначала заметим, что функция **f** возвращает куб переданного ей числа, а функция g – результат вычисления **2*n+3**
2) при некотором **i** работа цикла останавливается: это происходит при нарушении условия
   f(i)<g(k), то есть при выполнении обратного условия **f(i)≥g(k)**
3) в то же время на предыдущем шаге цикла (для **i-1**) условие его работы выполнялось, то есть
   f(i-1)<g(k), откуда получаем
   f(i-1)<g(k)≤f(i)
4) вспоминая, что $f(i)=i^3$, делаем вывод, что **g(k)** должно быть между кубами двух соседних чисел: $(i-1)^3 < g(k) \le i^3$
5) для заданного **k=10** находим **g(10)=2·10+3=23**, так что **i=3**:
   $(2)^3 < g(k)=23 \le 3^3$
6) осталось проверить, при каких k выполняется условие
   $8 < g(k)=2k+3 \le 27$
7) решая двойное неравенство относительно **k** получаем
   $2,5 < k \le 12$
8) таким образом, минимальное целое число, соответствующее условию – 3.

Ответ: 3.

Задача 21.8 Напишите в ответе число, равное количеству различных значений входной переменной k, при которых приведённая ниже программа выводит тот же ответ, что и при входном значении k = 10. Значение k = 10 также включается в подсчёт различных значений k.

```
var k, i : longint;
function f(n: longint) : longint;
begin
  f := n * n * n;
end;
begin
  readln(k);
  i := 1;
  while f(i) < k do
    i:= i+1;
  if f(i)-k <= k-f(i-1) then
       writeln(i)
  else writeln(i-1);
end.
```

Решение:

1) сначала заметим, что функция **f** возвращает куб переданного ей числа

2) после ввода **k** работает цикл, который увеличивает **i** до тех пор, пока значение куба **f(i)** не станет больше или равно **k** – тогда нарушится условие **f(i)<k** и цикл завершится
3) построим таблицу значений функции **f(i)** (кубов чисел):

i	f(i)	Цикл завершается для …
1	1	k=1
2	8	k=2,...,8
3	27	k=9,...,27

4) таким образом, при **k=10** цикл завершится при **i=3**
5) главная «новинка» – в условном операторе

```
if f(i)-k <= k-f(i-1) then
     writeln(i)
else writeln(i-1);
```

6) например, при **k=10** и **i=3** условие

```
f(i)-k <= k-f(i-1)
27-10 <= 10-8
17 <= 2
```

неверно, из-за этого выводится на экран не **i**, а **i-1**, то есть **2**
7) итак, нам нужно найти, сколько значений **k** дадут на выходе 2
8) посмотрим внимательно на условие в условном операторе, преобразуем его к виду

```
f(i)+f(i-1) <= 2k
```

9) тогда при выполнении обратного условия,

```
2k < f(i)+f(i-1)
k < (f(i)+f(i-1))/2
```

выводится число **i-1** вместо **i**
10) составим еще одну таблицу:

i	(f(i)+f(i-1))/2	Выводится i-1 для …	Выводится i для …
2	4,5	k=2,...,4	k=5,...,8
3	17,5	k=9,...,17	k=18,...,27

11) таким образом, в этой задаче нам подходят числа в диапазоне [5;17], всего их 17-5+1 = 13
12) Ответ: 13.

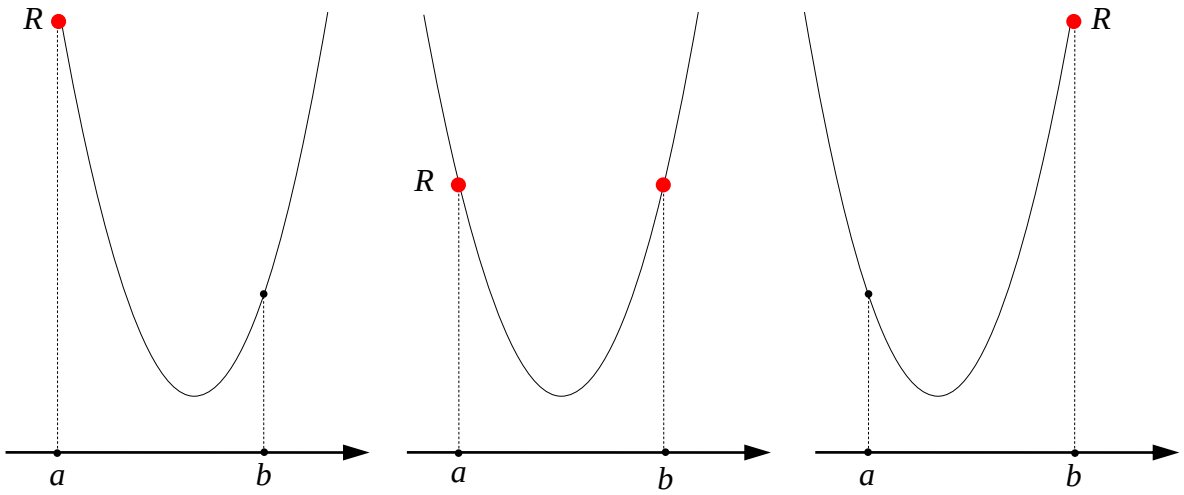
Задача 21.9. Определите, какое значение H нужно ввести, чтобы число, напечатанное в результате выполнения следующего алгоритма, было наименьшим.

```
var a,b,t,M,R,H :integer;
Function F(H, x: integer):integer;
begin
  F := 11*(x-H)*(x-H)+13;
end;
BEGIN
  readln(H);
  a := -10; b := 30;
  M := a; R := F(H, a);
  for t := a to b do begin
    if (F(H, t) > R) then begin
      M := t;
      R := F(H, t)
    end
  end;
```

```
  write(R)
END.
```

Решение:

1. заметим, величина **H** в программе не изменяется, то есть фактически выполняет роль константы; она передаётся в функцию и влияет на значение функции
2. что в программе есть цикл, в котором переменная **t** принимает последовательно все целые значения в интервале от **a** до **b**:
   ```
   for t:=a to b do begin
     ...
   end;
   ```
3. до начала цикла в переменную M записывается значение **a**, а в переменную **R** – значение функции в точке **a**:
   ```
   M:=a; R:=F(H,a);
   ```
4. внутри цикла есть условный оператор, в котором вычисляется значение функции **F(t)** и сравнивается со значением переменной **R**:
   ```
   if (F(H,t) > R)then begin
     M:=t;
     R:=F(H,t)
   end;
   ```
   если новое значение функции больше, чем значение **R**, в **R** записывается значение функции в точке **t**, а в переменной **M** запоминается само значение t (аргумент функции, соответствующий значению в **R**)
5. в результате анализа пп. 1-3 можно сделать вывод, что цикл ищет максимум функции **F(H,t)** на интервале от **a** до **b**
6. заметим, что выводится значение **R**, а величина **M** не выводится и не влияет на вычисление **R**, поэтому можно не обращать на неё внимания
7. функция **F** вычисляет значение
   ```
   F:=11*(x-H)*(x-H) + 13;
   ```
8. график этой эта функции – парабола с ветвями, направленными вверх (коэффициент при $x^2 = 11 > 0$)
9. вершина параболы находится в точке $x = H$, ветви идут симметрично влево и вправо вверх
10. при изменении H парабола двигается влево или вправо (но не вверх-вниз!)
11. итак, мы ищем максимальное значение квадратичной функции, и хотим, чтобы это значение было наименьшим
12. давайте подвигаем параболу в пределах отрезка $[a; b]$:



13. видно, что минимальное значение максимума будет тогда, когда вершина параболы будет расположена точно в середине отрезка [*a; b*]
14. отсюда требуемое значение *H* равно среднему арифметическому между *a* = –10 *и b* = 30:
    H = (–10 + 30) / 2 = 10
15. Ответ: 10.

Задача 21.10 Напишите в ответе число различных значений входной переменной k, при которых программа выдаёт тот же ответ, что и при входном значении k = 35. Значение k = 35 также включается в подсчёт различных значений k.

```
var k, i : longint;
function F(x: longint) : longint;
begin
  F:=2*x*x+3*x+2
end;
begin
  i := 15;
  readln(K);
  while (i> 0) and (F(i) > K) do
    i:=i-1;
  writeln(i)
end.
```

Решение (И. Тощенко)**:**

1) Вычислим значения функции F при i=1,2,3…
   i=0: *f*(0)=*2*
   i=1: *f*(1)=*7*
   i=2: *f*(2)=*16*
   i=3: *f*(3)=*29*
   i=4: *f*(4)=*46*
2) Заданное значение К попадает в отрезок [29;45].
3) Следовательно, всего 45-29+1=17 чисел.
4) ответ: 17.

Задача 21.10 *Напишите в ответе число различных значений входной переменной k, при которых программа выдаёт тот же ответ, что и при входном значении k = 64. Значение k = 64 также включается в подсчёт различных значений k.*

```
var k, i : longint;
function f(n: longint) : longint;
begin
```

```
  f := n * n
end;
begin
  readln(k);
  i := 12;
  while (i>0) and (f(i)>=k) do
    i := i-1;
  writeln(i)
end.
```

Решение:

1) заметим, что функция **F(x)** вычисляет квадрат переданного ей числа
2) в теле основной программы выполняется цикл с условием, который заканчивается, когда значение функции станет меньше **k**
3) на каждом шаге цикла уменьшается значение переменной **i**, начиная с 12; цикл также заканчивается, когда значение переменной **i** станет равно 0
4) после окончания цикла программа выводит значение переменной i.
5) итак, функция выводит первое натуральное значение **i**, квадрат которого меньше, чем введённое с клавиатуры значение переменной **k**
6) при **k = 64** программа выведет значение 7, поскольку это наибольшее натуральное число, квадрат которого меньше, чем 64
7) фактически нужно ответить на вопрос: сколько есть таких чисел **k**, которые меньше или равны $8^2 = 64$ и больше, чем $7^2 = 49$ (легко проверить, что при **k=65** программа выведет значение 8, в при **k=49** – значение 6)
8) в диапазоне [50;64] всего 64-50+1=15 чисел, это и есть правильный ответ.

Ответ: 15.

Задача 21.11. Определите, количество чисел K, для которых следующая программа выведет такой же результат, что и для K = 24:

```
var i, k: integer;
function F(x:integer):integer;
begin
  F:=x*x*x;
end;
begin
  i := 12;
  readln(K);
  while (i>0) and (F(i) > K) do
    i:=i-1;
  writeln(i);
end.
```

Решение:

заметим, что функция F(x) вычисляет куб переданного ей числа

перед началом цикла значение переменной i равно 12, в цикле оно уменьшается

цикл while останавливается, когда переменная i становится равна нулю или значение функции F(i) становится меньше или равно K

таким образом, в данной фактически требуется найти количество натуральных чисел в диапазоне [1..12], куб которых больше, чем K = 24

определим, у скольких чисел куб меньше, чем 24; это все числа, меньшие, чем $\sqrt[3]{24} \approx 2{,}88$, то есть, только числа 1 и 2; поэтому программа выведет 2 – первое число, куб которого меньше или равен 24

остаётся определить, когда программа выведет именно 2; это случится при всех K, при которых $2 \le \sqrt[3]{K} < 3$, то есть при $8 \le K < 27$; в этот диапазон входит 27-8 = 19 чисел

Ответ: 19.

Задача 21.12. Определите, количество чисел K, для которых следующая программа выведет такой же результат, что и для K = 24:

```
var i, k: integer;
function F(x:integer):longint;
begin
  if x = 1 then
       F:=1
  else F:=x*F(x-1);
end;
begin
  i := 12;
  readln(K);
  while (i>0) and (F(i) > K) do
    i:=i-1;
  writeln(i);
end.
```

Решение:

заметим, что рекурсивная функция F(x) вычисляет факториал переданного ей числа, то есть произведение x!=1·2·3·...·(x-1) ·x

повторяя рассуждения предыдущей задачи, определяем, что функция выведет количество натуральных чисел, факториалы которых меньше или равны K

выпишем факториалы первых натуральных чисел:

1! = 1, 2! = 2, 3! = 6, 4! = 24, 5! = 120, 6! = 720, …

из этого ряда факториалы первых четырех чисел меньше или равны 24, поэтому при K=24 функция выведет число 4

программа выведет именно 4 при всех K, при которых $4! = 24 \le K < 5! = 120$, то есть при $24 \le K < 120$; в этот диапазон входит 120-24 = 96 чисел

Ответ: 96.

. **Задача 21.13** Определите, какое число будет напечатано в результате выполнения следующего алгоритма:

```
var a, b, t, N, P :integer;
Function F(x: integer):integer;
begin
  F := 16*(9-x)*(9-x)+127;
end;
BEGIN
  a := -25; b := 25;
  P := 130;
  N := 0;
```

```
    for t := a to b do begin
      if (F(t) > P) then begin
        N := N+1;
      end;
    end;
    write(N);
END.
```

Решение:

заметим, что в конце работы программы на экран выводится значение переменной N

в программе значение N сначала обнуляется, а затем на каждом шаге цикла увеличивается на 1 при условии, что значение функции F(t) больше значения P = 130; таким образом, N – это счётчик точек с целочисленными значениями на отрезке [-25;25], в которых значение функции больше, чем 130

график функция 16*(9-x)*(9-x)+127 – парабола с ветвями вверх, минимальное значение в точке x = 9 равно 127

значение функции при x = 8 и x = 10 (рядом с точкой минимума) равны 16+127 = 143, поэтому только в одной точке x = 9 не выполняется условие F(t) > P

всего на интервале [-25;25] есть 51 точка с целочисленными координатами; во всех, за исключением одной условие F(t) > P выполняется, то есть счётчик увеличивается на 1

Ответ: 50.

Задача 21.13. Определите, какое число будет напечатано в результате выполнения следующего алгоритма:

```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:=(x*x-4)*(x*x-4)+6;
end;
BEGIN
  a:=-10; b:=10;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)<R)then begin
      M:=t;
      R:=F(t);
    end;
  end;
  write(M+6);
END.
```

Решение:

заметим, что в программе есть цикл, в котором переменная t принимает последовательно все целые значения в интервале от a до b:

```
for t:=a to b do begin
  ...
end;
```

до начала цикла в переменную M записывается значение a, а в переменную R – значение функции в точке a:

M:=a; R:=F(a);

внутри цикла есть условный оператор, в котором вычисляется значение функции F(t) и сравнивается со значением переменной R:

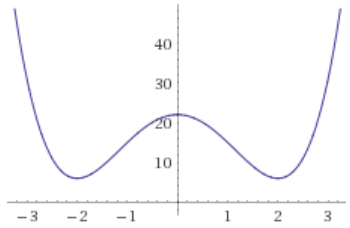
```
if (F(t)<R)then begin
  M:=t;
  R:=F(t);
end;
```

если новое значение функции меньше, чем значение R, в R записывается значение функции в точке t, а в переменной M запоминается само значение t (аргумент функции, соответствующий значению в R)

в результате анализа пп. 1-3 можно сделать вывод, что цикл ищет минимум функции F(t) на интервале от a до b, и после выполнения цикла в переменной M оказывается значение аргумента t, при котором функция достигает минимума на заданном интервале (здесь это интервал [-10, 10])

функция, которая используется в программе, – это квадратичная парабола:

$y = (x^2 - 4)^2 + 6$, её ветви направлены вверх (коэффициент при x^4 положительный, равен 1); она имеет два минимума в точках $x = -2$ и $x = 2$



обе точки минимума находятся на отрезке [-10;10], поэтому программа найдёт одну из этих точек; вопрос: какую именно?

для квадратичной параболы обе точки минимума имеют одинаковую y-координату, а запоминание новой точки минимума происходит только тогда, когда только что вычисленное значение F(t) станет строго меньше, чем хранящееся в переменной R:

```
if (F(t) < R)then begin
  M:=t;
  R:=F(t);
end;
```

поэтому в точке второго минимума $x = 2$ никаких изменений не произойдет, и в переменной M останется значение «–2»; таким образом, будет найдет первый минимум

обратим внимание, что на экран выводится не M, а M+6, поэтому результат будет равен (-2)+6=4

Ответ: 4.

Задача 21.14. Определите, какое число будет напечатано в результате выполнения следующего алгоритма:

```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:=4*(x-1)*(x-3);
end;
```

```
BEGIN
  a:=-20; b:=20;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)<R)then begin
      M:=t;
      R:=F(t);
    end;
  end;
  write(M);
END.
```

Решение (способ 1, ручная прокрутка, перебор):

1. заметим, что в программе есть цикл, в котором переменная t принимает последовательно все целые значения в интервале от a до b:
```
for t:=a to b do begin
...
end;
```
2. до начала цикла в переменную M записывается значение a, а в переменную R – значение функции в точке a:
M:=a; R:=F(a);
3. внутри цикла есть условный оператор, в котором вычисляется значение функции F(t) и сравнивается со значением переменной R:
```
if (F(t)<R)then begin
M:=t;
R:=F(t);
end;
```
4. если новое значение функции меньше, чем значение R, в R записывается значение функции в точке t, а в переменной M запоминается само значение t (аргумент функции, соответствующий значению в R)
5. в результате анализа пп. 1-3 можно сделать вывод, что цикл ищет минимум функции F(t) на интервале от a до b, и после выполнения цикла в переменной M оказывается значение аргумента t, при котором функция достигает минимума на заданном интервале (здесь это интервал [-20, 20])
6. функция F вычисляет значение
7. `F:=4*(x-1)*(x-3);`
8. перебираем все значения t от a до b, и для каждого вычисляем соответствующее значение функции:

t	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0
F	1932	1760	1596	1440	1292	1152	1020	896	780	672	572	480	396	320	252	192	140	96	60	32	12

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	0	-4	0	12	32	60	96	140	192	252	320	396	480	572	672	780	896	1020	1152	1292

1. по таблице находим, что минимальное значение **-4** достигается при **t=2**
2. таким образом, ответ: 2.

Возможные проблемы:
- заполнение таблицы, особенно при большом интервале, очень трудоемко, велика возможность ошибки

Решение (способ 2, математический анализ):

1) повторяя рассуждения пп. 1-5 из предыдущего способа решения, находим, что программа ищет значение **t**, при котором функция **F(t)** принимает минимальное значение на интервале от **a** до **b**.

2) запишем функцию в виде квадратного трёхчлена:
$F(x) = 4(x-1)(x-3) = 4(x^2 - 4x + 3)$
3) график этой функции – парабола, оси которой направлены вверх, поэтому функция имеет минимум
4) найдем абсциссу точки минимума, которая совпадает с абсциссой точки минимума функции

$$F_1(x) = x^2 - 4x + 3 \quad \Rightarrow \; x_{\min} = \frac{-b}{2a} = \frac{-(-4)}{2 \cdot 1} = 2$$

Ответ: 2.

Решение (способ 3, математический анализ, свойства параболы):

1) повторяя рассуждения пп. 1-5 из первого способа решения, находим, что программа ищет значение **t**, при котором функция **F(t)** принимает минимальное значение на интервале от **a** до **b**.
2) заданная функция $F(x) = 4(x-1)(x-3)$ имеет корни в точках $x_1 = 1, \; x_2 = 3$
3) график этой функции – парабола, оси которой направлены вверх (коэффициент при x^2 равен 4 > 0), поэтому функция имеет минимум
4) парабола симметрична относительно вертикальной прямой, проходящей через вершину, поэтому абсцисса вершины – это среднее арифметическое корней:

$$x_{\min} = \frac{1+3}{2} = 2$$

Ответ: 2.

Задача 21.15 Определите, какое число будет напечатано в результате выполнения следующего алгоритма:

```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:=x*x + 4*x + 8;
end;
BEGIN
  a:=-10; b:=10;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)> R)then begin
      M:=t;
      R:=F(t);
    end;
  end;
  write(R);
END.
```

Решение:

Рассуждая так же, как и в предыдущем примере, можно показать, что программа ищет наибольшее значение функции F(t) на интервале от a до b

Заметим, что выводится не абсцисса, а именно это найденное наибольшее значение функции:

Write(R);

График заданной функции $F(x) = x^2 + 4x + 8$ – это парабола, ветви которой направлены вверх, то есть она имеет точку минимума, но не точку максимума

Поэтому нужно проверить значения функции на концах отрезка и выбрать из них наибольшее
При t=-10 получаем F(t)=68
При t=10 получаем F(t)=148
Таким образом, ответ: 148.
Задача 21.16 Определите, какое число будет напечатано в результате выполнения следующего алгоритма:

```
Var a,b,t,M,R:integer;
Function F(x:integer):integer;
begin
  F:=4*(x-1)*(x-3);
end;
BEGIN
  a:=-20; b:=0;
  M:=a; R:=F(a);
  for t:=a to b do begin
    if (F(t)<R)then begin
      M:=t;
      R:=F(t);
    end;
  end;
  write(M);
END.
```

Решение:

1) рассуждая так же, как и в примере 1, определяем, что программа ищет значение **t**, при котором функция **F(t)** принимает *минимальное* значение на интервале от **a** до **b**.
2) запишем функцию в виде квадратного трёхчлена:
   $F(x) = 4(x-1)(x-3) = 4(x^2 - 4x + 3)$
3) график этой функции – парабола, оси которой направлены вверх, поэтому функция имеет минимум
4) найдем абсциссу точки минимума, которая совпадает с абсциссой точки минимума функции
   $F_1(x) = x^2 - 4x + 3 \quad \Rightarrow x_{\min} = \frac{-b}{2a} = \frac{-(-4)}{2 \cdot 1} = 2$
1) однако это значение не входит в интервал [-20; 0], поэтому нужно проверить значения функции на концах отрезка и выбрать из них наименьшее; ответом будет соответствующее значение **t**.
2) при **t=-20** получаем **F(-20)=4*(-21)*(-23)=1932**
3) при **t=0** получаем **F(0)= 4*(-1)*(-3)=12**, это значение меньше, чем **F(-20)**, поэтому минимум на заданном интервале достигается при **t=0**

Ответ: 0

Задача 22 (повышенный уровень, время –7 мин)

Тема: динамическое программирование.

Что нужно знать:

- динамическое программирование – это способ решения сложных задач путем сведения их к более простым задачам того же типа
- с помощью динамического программирования решаются задачи, которые требуют полного перебор вариантов:
  - «подсчитайте количество вариантов…»
  - «как оптимально распределить…»
  - «найдите оптимальный маршрут…»

динамическое программирование позволяет ускорить выполнение программы за счет использования дополнительной памяти; полный перебор не требуется, поскольку запоминаются решения всех задач с меньшими значениями параметров

Задача 22.1 *(демо-вариант 2018 г.).*

Исполнитель М17 преобразует число на экране. У исполнителя есть три команды, которым присвоены номера:

1. Прибавить 1

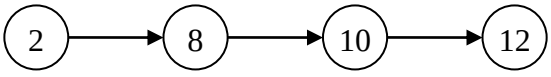
2. Прибавить 2

3. Умножить на 3

Первая команда увеличивает число на экране на 1, вторая – увеличивает его на 2, а третья – умножает его на 3. Программа для исполнителя М17 – это последовательность команд. Сколько существует программ, для которых при исходном числе 2 результатом является число 12 и при этом траектория вычислений содержит числа 8 и 10?

Решение:

1) запишем рекуррентную формулу для вычисления K_N – количества возможных программ для получения числа N из некоторого начального числа:
   $K_N = K_{N-1} + K_{N-2}$, если N не делится на 3
   $K_N = K_{N-1} + K_{N-2} + K_{N/3}$, если N делится на 3
2) все допустимые программы можно разбить на 3 части:
   – переход от 2 до 8
   – переход от 8 до 10
   – переход от 10 до 12



3) обозначим через $K_{a\to b}$ количеств возможных программ получения числа b из числа a
4) очевидно, что $K_{a\to b} = K_{a\to c} \cdot K_{c\to b}$ для любого c, такого что $a < c < b$
5) поэтому $K_{2\to 12} = K_{2\to 8} \cdot K_{8\to 10} \cdot K_{10\to 12}$
6) вычисляем эти значения отдельно стандартным способом по рекуррентным формулам п. 1:

N	2	3	4	5	6	7	8
K_N	1	1	2	3	6	9	15

N	8	9	10
K_N	1	1	2

10	11	12
1	1	2

7) и перемножаем: 15 · 2 · 2 = 60

Ответ: 60.

Задача 22.2

Исполнитель Июнь15 преобразует число на экране. У исполнителя есть две команды, которым присвоены номера:

*1. **Прибавить 1***

*2. **Умножить на 2***

Первая команда увеличивает число на экране на 1, вторая умножает его на 2. Программа для исполнителя Июнь15 – это последовательность команд. Сколько существует программ, для которых при исходном числе 2 результатом является число 29 и при этом траектория вычислений содержит число 14 и не содержит числа 25?

Решение:

у нас в задании две особые точки – числа 14 (через которое должна проходить траектория) и 25 (а сюда она попасть НЕ должна)

сначала, так же, как и в задачах, рассмотренных ниже, составляем рекуррентную формулу, по которой будем вычислять количество K_N обозначить количество разных программ для получения числа N из начального числа:

число N могло быть получено одной из двух операций:

увеличением на 1 числа N-1;

умножением на 2 числа N/2 (только для N, которые делятся на 2);

$K_N = K_{N-1}$ для нечётных чисел

$K_N = K_{N-1} + K_{N/2}$ для чётных чисел

для начального числа 2 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды; $K_1 = 1$.

составляем таблицу до первой особой точки – числа 14:

N	2	3	4	5	6	7	8	9	10	11	12	13	14
K_N	1	1	2	2	3	3	5	5	7	7	10	10	13

поскольку число 14 должно обязательно войти в траекторию, начинаем составлять вторую часть таблицы (до второй контрольной точки, 25) с этого числа заново, считая, что все ячейки для меньших чисел – нулевые

N	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
K_N	13	13	13	13	13	13	13	13	13	13	13	0				

поскольку траектория не может проходить через 25, для N = 25 принимаем KN = 0 (в таблице эта ячейка выделена красным цветом)

дальше заполняем оставшиеся ячейки второй части таблицы обычным способом (см. задачи ниже):

N	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
K_N	13	13	13	13	13	13	13	13	13	13	13	0	0	0	13	13

Ответ: 13.

Задача 22.3

У исполнителя Удвоитель две команды, которым присвоены номера:

1. Прибавить 1

2. Умножить на 2

Первая команда увеличивает число на экране на 1, вторая умножает его на 2. Программа для исполнителя Удвоитель – это последовательность команд. Сколько существует программ, преобразующих число 4 в число 24, предпоследней командой которых является команда «1»?

Решение:

итак, мы знаем предпоследнюю команду – 1, при этом последняя команда может быть любая – 1 или 2

выходит, что нужно получить количество всех программ вида «*11» и «*12», где звёздочка обозначает любые команды

если программа заканчивается на «11», то до выполнения цепочки «11» у нас было число

24 – 1 – 1 = 22; поэтому нужно найти число программ для преобразования 4 в 22

для начального числа 1 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды; если через K_N обозначить количество разных программ для получения числа N из начального числа 1, то $K_1 = 1$.

теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_{N-1}$, то есть с решениями таких же задач для меньших N

число N могло быть получено одной из двух операций:

увеличением на 1 числа N-1;

умножением на 2 числа N/2 (только для N, которые делятся на 2, и таких, что N/2 ≥ 4);

$K_N = K_{N-1}$ для нечётных чисел

$K_N = K_{N-1} + K_{N/2}$ для чётных чисел, таких, что N/2 ≥ 4

составляем таблицу:

N	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
K_N	1	1	1	1	2	2	3	3	4	4	5	5	7	7	9	9	12	12	15

теперь рассматриваем случай, когда программа заканчивается на «12», это значит, что до выполнения цепочки «12» у нас было число (24/ 2) – 1 = 11; поэтому нужно найти число программ для преобразования 4 в 11, берём его из таблицы: 3

ответ к задаче – сумма двух значений, выделенных жёлтым маркером: 15 + 3 = 18, поскольку мы рассмотрели все варианты программ, в которых предпоследняя команда **– 1**

Ответ: 18.

Задача 22.4

Исполнитель Июнь15 преобразует число на экране. У исполнителя есть две команды, которым присвоены номера:

1. Прибавить 1

2. Умножить на 2

Первая команда увеличивает число на экране на 1, вторая умножает его на 2. Программа для исполнителя Июнь15 – это последовательность команд. Сколько существует программ, для которых при исходном числе 1 результатом является число 21 и при этом траектория вычислений содержит число 10? Траектория вычислений программы – это последовательность результатов
выполнения всех команд программы. Например, для программы 121 при исходном числе 7 траектория будет состоять из чисел 8, 16, 17.

Решение (вариант 1):

заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)

для начального числа 1 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды; если через K_N обозначить количество разных программ для получения числа N из начального числа 1, то $K_1 = 1$.

теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_{N-1}$, то есть с решениями таких же задач для меньших N

число N могло быть получено одной из двух операций:

увеличением на 1 числа N-1;

умножением на 2 числа N/2 (только для N, которые делятся на 2);

$K_N = K_{N-1}$ для нечётных чисел

$K_N = K_{N-1} + K_{N/2}$ для чётных чисел

поскольку траектория должна проходить через число 10, сначала выясняем, сколькими способами можно получить 10 из 1, а затем будем считать, сколько есть способов получить 21 из 10

заполняем таблицу от 1 до 10 по полученным формулам:

N	1	2	3	4	5	6	7	8	9	10
K_N	1	2	2	4	4	6	6	10	10	14

второй этап – определяем таким же образом (и по таким же формулам!), сколько есть способов получить конечное число 21 из 10, только левую часть таблицы (от 1 до 10) мы уже не рассматриваем:

N	10	11	12	13	14	15	16	17	18	19	20	21
K_N	14	14	14	14	14	14	14	14	14	14	28	28

Ответ: 28.

Решение (вариант 2, А.Н. Носкин):

первый этап (п. 1-6) такой же, как и в первом варианте (см. выше);

на втором этапе используем такую идею: если мы знаем количество команд, с помощью которых из начального числа 1 можно получить 10 и определим количество ко-

манд, с помощью которых из 10 можно получить конечное значение 21, останется только перемножить эти два числа – это и будет ответ

составляем таблицу для получения 21 из 10, используя те же рекуррентные формулы:

N	10	11	12	13	14	15	16	17	18	19	20	21
K_N	1	1	1	1	1	1	1	1	1	1	2	2

результат – 14 × 2 = 28

Ответ: 28.

Задача 22.5.

Исполнитель Калькулятор преобразует число, записанное на экране. У исполнителя три команды, которым присвоены номера:

1. прибавь 1

2. прибавь 2

3. прибавь следующее

Первая из них увеличивает число на экране на 1, вторая увеличивает это число на 2, а третья прибавляет к числу на экране число, большее на 1 (к числу 3 прибавляется 4, к числу 9 прибавляется 10 и т. д.). Программа для исполнителя Калькулятор– это последовательность команд. Сколько есть программ, которые число 2 преобразуют в число 10?

Решение (1 способ, составление таблицы):

1. заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)
2. для начального числа 2 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды; если через K_N обозначить количество разных программ для получения числа N из начального числа 2, то $K_2 = 1$.
3. теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_{N-1}$, то есть с решениями таких же задач для меньших N
4. число N могло быть получено одной из трёх операций сложения:

увеличением на 1 числа N-1;

увеличением на 2 числа N-2;

из некоторого числа X увеличением на X+1 (следующее число), так что N = X + X + 1, откуда X = (N – 1) / 2; так могут быть получены только нечетные числа;

поэтому

$K_N = K_{N-1} + K_{N-2}$ для чётных чисел

$K_N = K_{N-1} + K_{N-2} + K_{(N-1)/2}$ для нечётных чисел

остается по этой формуле заполнить таблицу для всех значений от 2 до 10:

N	2	3	4	5	6	7	8	9	10
K_N	1	1	2	4	6	11	17	30	47

Ответ – 47.

Задача 22.6

Исполнитель Май4 преобразует число, записанное на экране. У исполнителя три команды, которым присвоены номера:

1. прибавь 1

2. прибавь 2

3. прибавь 4

Первая из них увеличивает число на экране на 1, вторая увеличивает это число на 2, а третья – на 4. Программа для исполнителя Май4 – это последовательность команд. Сколько есть программ, которые число 21 преобразуют в число 30?

Решение (1 способ, составление таблицы):

заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)

все числа, меньшие начального числа 21, с помощью этого исполнителя получить нельзя, для них количество программ будет равно 0

для начального числа 21 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды; если через K_N обозначить количество разных программ для получения числа N из начального числа 21, то $K_{21} = 1$.

теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_{N-1}$, то есть с решениями таких же задач для меньших N

любое число N > 21 могло быть получено одной из трёх операций сложения соответственно из чисел N-1, N-2 и N-4, поэтому

$K_N = K_{N-1} + K_{N-2} + K_{N-4}$

остается по этой формуле заполнить таблицу для всех значений от 21 до 30:

N	21	22	23	24	25	26	27	28	29	30
K_N	1	1	2	3	6	10	18	31	55	96

Ответ – 96.

Задача 22.6

У исполнителя Утроитель две команды, которым присвоены номера:

1. прибавь 1

2. умножь на 3

Первая из них увеличивает число на экране на 1, вторая – утраивает его. Программа для Утроителя – это последовательность команд. Сколько есть программ, которые число 1 преобразуют в число 20?

Решение (1 способ, составление таблицы):

заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)

начнем с простых случаев, с которых будем начинать вычисления: для чисел 1 и 2, меньших, чем 3, существует только одна программа, состоящая только из команд сложения; если через K_N обозначить количество разных программ для получения числа N из 1, то $K_1 = K_2 = 1$.

теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_{N-1}$, то есть с решениями таких же задач для меньших N

если число N не делится на 3, то оно могло быть получено только последней операцией сложения, поэтому $K_N = K_{N-1}$

если N делится на 3, то последней командой может быть как сложение, так и умножение

поэтому для получения K_N нужно сложить K_{N-1} (количество программ с последней командой сложения) и $K_{N/3}$ (количество программ с последней командой умножения). В итоге получаем:

если N не делится на 3: $K_N = K_{N-1}$

если N делится на 3: $K_N = K_{N-1} + K_{N/3}$

остается заполнить таблицу для всех значений от 1 до N:

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
K_N	1	1	2	2	2	3	3	3	5	5	5	7	7	7	9	9	9	12	12	12

Заметим, что количество вариантов меняется только в тех столбцах, где N делится на 3, поэтому из всей таблицы можно оставить только эти столбцы:

N	1	3	6	9	12	15	18	21
K_N	1	2	3	5	7	9	12	15

заданное число 20 попадает в последний интервал (от 18 до 21), поэтому …

Ответ – 12.

Задача 22.7

У исполнителя Калькулятор две команды, которым присвоены номера:

1. прибавь 1

2. увеличь вторую с конца цифру на 1

Первая из них увеличивает число на экране на 1, вторая – увеличивает на 1 число десятков. Если перед выполнением команды 2 вторая с конца цифра равна 9, она не изменяется. Программа для Калькулятора – это последовательность команд.

Сколько есть программ, которые число 15 преобразуют в число 28?

Решение (1 способ, составление таблицы):

заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)

при заданных командах очередное число N может быть получено двумя способами:

увеличением на 1 (для всех чисел, больших начального числа)

увеличением числа десятков на 1 (то есть, фактически командой «+10») – для всех чисел, больших или равных 25; например, число 24 не может быть получено этой командой (14 + 10 = 24), потому что число 14 меньше, чем начальное значение 15

таким образом, рекуррентные формулы принимают вид

$K_N = K_{N-1}$ для всех чисел, меньших, чем 25

$K_N = K_{N-1} + K_{N-10}$ для чисел, больших или равных 25

других способов получения числа с помощью исполнителя с заданными командами нет, то есть мы таким образом рассматриваем все возможные программы

начальное значение: $K_{15} = 1$ (число 15 можно получить единственной пустой программой)

далее заполняем таблицу:

N	15	16	17	18	19	20	21	22	23	24	25	26	27	28
K_N	1	1	1	1	1	1	1	1	1	1	2	3	4	5

Ответ: 5

Задача 22.8

У исполнителя Калькулятор две команды, которым присвоены номера:

1. прибавь 1

2. увеличь две младшие цифры на 1

Первая из них увеличивает число на экране на 1, вторая – увеличивает на 1 число десятков и число единиц. Если перед выполнением команды 2 какая-либо из двух младших цифр равна 9, она не изменяется. Программа для Калькулятора – это последовательность команд.

Сколько есть программ, которые число 23 преобразуют в число 48?

Решение (1 способ, составление таблицы):

заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)

при заданных командах очередное число N может быть получено двумя способами:

увеличением на 1 (для всех чисел, больших начального числа)

увеличением обеих цифр на 1 в результате выполнения команды 2 (то есть, фактически командой «+11») – для всех чисел, больших или равных 23 + 11 = 34, которые НЕ оканчиваются на 0;

увеличением только младшей цифры на 1 в результате выполнения команды 2 (то есть, фактически командой «+1») – для всех чисел от 91 до 99, но в нашем диапазоне (23..48) таких нет

увеличением только старшей цифры на 1 в результате выполнения команды 2 (то есть, фактически командой «+10») – для всех чисел, больших 34 и имеющих 9 на конце; в нашем случае под этот вариант подходит только число 39

таким образом, рекуррентные формулы принимают вид

$K_N = K_{N-1}$ для всех чисел, меньших, чем 34, а также для всех чисел, оканчивающихся на 0

$K_N = K_{N-1} + K_{N-11}$ для чисел, больших или равных 34, кроме 39

$K_N = K_{N-1} + K_{N-11} + K_{N-10}$ для числа 39

других способов получения числа с помощью исполнителя с заданными командами нет, то есть мы таким образом рассматриваем все возможные программы

начальное значение: $K_{23} = 1$ (число 23 можно получить единственной пустой программой)

далее заполняем таблицу:

N	23	…	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
K_N	1	…	1	2	3	4	5	6	8	8	9	10	11	12	14	17	21	26

здесь многоточия означают, что для всех чисел от 23 до 33 включительно количество программ равно 1;

например, для числа 47 количество программ вычисляется как

$K_{47} = K_{46} + K_{36} = 17 + 4 = 21$

а для числа 39 –как

$K_{39} = K_{38} + K_{28} + K_{29} = 6 + 1 + 1 = 8$

Ответ: 26

27(С4) (высокий уровень, время – 55 мин)

Тема: Обработка данных, вводимых в виде символьных строк (написать программу средней сложности из 30-50 строк) или последовательности чисел.

Что нужно знать:

символьная строка – это цепочка символов, которая может обрабатываться как единое целое

для обращения к символу с номером i строки s используется запись s[i], это говорит о том, что строка – особый вариант массива, в котором хранятся символы

знак сложения при работе с символьными строками означает сцепку, объединение двух строк в одну (добавление второй строки в конец первой), например:

```
s := '123' + '456'; { получили '123456' }
```

с помощью функции Ord можно получить код символа; цифры имеют коды от 48 (цифра 0) до 57 (цифра 9), например

```
k := Ord('1');   { получили 49 }
```

то же самое можно сделать с помощью преобразования типа (привести char к integer)

```
k := integer('1'); { получили 49 }
```

с помощью функции Chr можно сделать обратный переход: получить символ по его коду, например

```
c := Chr(49);  { получили символ '1' }
```

то же самое можно сделать с помощью преобразования типа (привести integer к char)

```
c := char(49); { получили символ '1' }
```

для работы со строками в наиболее распространенных Паскаль-средах (Turbo Pascal, Borland Pascal, PascalABC, среда АЛГО) используют стандартные функции (здесь s – это переменная типа string, символьная строка; n и r – целые переменные)

`n := Length(s);`	записать длину строки s в целую переменную n
`s1 := Copy(s, 2, 5);`	записать в символьную строку s1 подстроку строки s, которая начинается с символа с номером 2 и состоит из 5 символов (важно – не со 2-го по 5-ый символ!)
`n := Pos('Вася', s);`	записать в целую переменную n номер символа, с которого в строке s начинается подстрока 'Вася' (если ее нет, в переменную n записывается 0); так же можно искать отдельные символы (важно: сначала указываем, что ищем, а потом – где)
`n := StrToInt(s);`	преобразовать строку s в целое число и записать результат в переменную n (PascalABC, Delphi)

и процедуры

`Delete(s, 2, 5);`	удалить из строки s 5 символов, начиная со второго
`Insert('Вася', s, 3);`	вставить в строку s фрагмент 'Вася', начиная с третьего символа (между 2-м и 3-м)
`Val(s, n, r);`	преобразовать строку s в целое число и записать результат в переменную n; если при этом произошла ошибка, в переменной r будет номер ошибочного символа, если все нормально – ноль

структура (в Паскале она называется «запись», record) – это сложный тип данных, который может включать в себя несколько элементов – полей; поля могут иметь различный тип

записи в Паскале объявляются с помощью ключевого слова record; в простейшем случае можно выделить память под одну запись так:

```
var x: record
     name: string;
     code: integer;
    end;
```

эта запись состоит из двух полей: символьной строки name и целого числа code
записи очень удобны для работы, когда все данные в целом представляют собой единый блок информации, например, данные об ученике; если не использовать записи, было бы нужно выделять в памяти отдельно символьную строку и отдельно целую переменную, причем эти данные внешне были бы никак не связаны, поэтому программа с записями часто получается логичнее и понятнее как для автора, так и для того, кто будет в ней разбираться
для обращения к полям записи используют точку, например x.name означает «поле name записи x»
можно сразу объявить массив записей:

```
var Info: array[1..100] of record
     name: string;
     code: integer;
    end;
```

это 100 одинаковых записей, имеющих общее имя Info и расположенных в памяти рядом; в каждой структуре есть поля name и code; чтобы работать с полями записи с номером k используют обращения вида Info[k].name и Info[k].code

Сложность алгоритмов:

- обозначение $O(N)$ говорит о том, что при увеличении в 2 раза размера массива данных количество операций тоже увеличивается примерно в 2 раза (для больших N)
- сложность $O(N)$ имеет алгоритм с одним или несколькими простыми (не вложенными!) циклами в каждом из которых выполняется N шагов (как при поиске минимального элемента)
- количество операций для алгоритма, имеющего сложность $O(N)$, вычисляется по формуле $p = a \cdot N + b$, где a и b – некоторые постоянные
- если в одном алгоритме решения задачи используется несколько циклов от 1 до N, а во втором – только один цикл, то алгоритм с одним циклом, как правило, эффективнее (хотя оба алгоритма имеют сложность $O(N)$, постоянная a в каждом случае своя, для алгоритма с несколькими циклами она будет больше)
- для алгоритма, имеющего сложность $O(N^2)$, количество операций пропорционально квадрату размера массива, то есть, если N увеличить в 2 раза, то количество операций увеличивается примерно в 4 раза (например, в программе используется два вложенных цикла, в каждом из которых N шагов); сложность $O(N^2)$ имеют простые способы сортировки массивов: метод «пузырька», метод выбора
- при больших N функция $f_1(N) = a_1 N^2$ растет значительно быстрее, чем $f_2(N) = a_2 N$, поэтому алгоритм, имеющий сложность $O(N^2)$ всегда менее эффективен, чем алгоритм сложности $O(N)$
- иногда встречаются алгоритмы сложности $O(N^3)$ (три вложенных цикла от 1 до N), при больших N они работают медленнее, чем любой алгоритм сложности $O(N^2)$, то есть, менее эффективны

для многих задач известны только алгоритмы экспоненциальной сложности, когда размер массива входит в показатель степени, например $O(2^N)$, для больших N такие задачи не решаются за приемлемое время (например, «взламывание» шифров)

Задача 27.1На вход программе подаются сведения о номерах школ учащихся, участвовавших в олимпиаде. В первой строке сообщается количество учащихся N, каждая из следующих N строк имеет формат:
<Фамилия> <Инициалы> <номер школы>
где <Фамилия> – строка, состоящая не более чем из 20 символов, <Инициалы> – строка, состоящая из 4-х символов (буква, точка, буква, точка), <номер школы> – не более чем двузначный номер. <Фамилия> и <Инициалы>, а также <Инициалы> и <номер школы> разделены одним пробелом. Пример входной строки:
Иванов П.С. 57
Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет выводить на экран информацию, из какой школы было меньше всего участников (таких школ может быть несколько). При этом необходимо вывести информацию только по школам, пославшим хотя бы одного участника. Следует учитывать, что N>=1000.

Как правильно понимать условие?

1. на первый вопрос – как именно вводятся данные – находим ответ в самом начале условия: вроде бы «дежурная» фраза «на вход программе подаются…» означает, что данные нужно читать не из файла, а со стандартного входного потока; это, в свою очередь, значит, что можно использовать привычные операторы **read** (**readln**), предполагая, что кто-то вводит эти данные с клавиатуры вручную[3]
2. итак, сначала вводится количество записей в файле **N**, а затем **N** строк с информацией; заметим, что из всей этой информации нас интересует (в каждой строке) только номер школы, остальное можно просто отбрасывать
3. номер школы стоит после второго пробела в строке
4. «*<номер школы> – не более чем двузначный номер*» – крайне важная информация; собственно, только она и позволяет найти хорошее решение задачи; это значит, что школ не более 99!
5. что означает выражение «как можно более эффективная программа»?
6. прежде всего, данные читаются только один раз, за один проход, нельзя «вернуться» и прочитать что-то вновь
7. в программе не выполняются никакие лишние действия
8. используемые алгоритмы имеют минимальную сложность (см. выше)
9. расходуется минимальный возможный объем памяти; например, чтобы найти количество отрицательных элементов массива, не нужно вводить второй массив; если нам достаточно держать в памяти одну введенную строку, не нужно одновременно хранить все прочитанные строки
10. зачем нужно уточнение «*N>=1000*»? этим авторы задачи намекают на то, что не нужно считывать все данные в оперативную память, а потом уже их обрабатывать; основная обработка должна быть сделана сразу, в том же цикле, где читаются вход-

ные данные мы будем считать, что в исходных данных нет ошибок (так принято на олимпиадах и экзаменах), иначе обработка разнообразных ошибок будет составлять основную часть программы

Решение:

1) по условию, единственная информация, которая нам нужна в итоге для вывода результата – это количество участников по каждой школе

2) так как номер школы состоит (по условию!) не более, чем из двух цифр, всего может быть не более 99 школ (с номерами от 1 до 99)

3) поэтому можно ввести массив **C** из 99 элементов; для всех **k** от 1 до 99 элемент **C[k]** будет ячейкой-счетчиком, в которой накапливается число участников от школы с номером **k**; сначала во все элементы этого массива записываются нуль (обнуление счетчиков):

```
for k:=1 to 99 do C[k]:=0;
```

во многих системах программирования на Паскале все глобальные переменные автоматически обнуляются, и таким образом, этот цикл ничего не дает; однако на всякий случай нужно продемонстрировать эксперту, который будет проверять часть **C** вашей работы, что вы понимаете суть дела («счетчик необходимо сначала обнулить»)

4) основной цикл обработки вводимых строк можно записать на псевдокоде так:

```
for i:=1 to N do begin
 { читаем очередную строку }
 { определяем номер школы k }
 C[k] := C[k] + 1; { увеличиваем счетчик k-ой школы }
end;
```

5) поскольку данные вводятся в виде символьной строки, нужно выделить в памяти переменную **s** типа **string**

6) для чтения очередной строки будем использовать оператор **readln**

7) остается понять, как выделить из строки номер школы; по условию он закодирован в последней части строки, после второго пробела; значит, нужно найти этот второй пробел, вырезать из строки весь «хвост» после этого пробела, и преобразовать его из символьного формата в числовой

8) чтобы найти первый пробел и «отрезать» первую часть строки с этим пробелом, можно использовать команды

```
p := Pos(' ', s);
s := Copy(s, p+1, Length(s)-p);
```

первая команда определяет номер первого пробела и записывает его в целую переменную **p**, в вторая – записывает в строку **s** весь «хвост», стоящий за этим пробелом, начиная с символа с номером **p+1**; длина хвоста равна **Length(s)-p**, где **Length(s)** – длина строки;

9) поскольку нас интересует часть после **второго** пробела, эти две строчки нужно повторить два раза, в результате в переменной **s** окажется символьная запись номера школы;

10) заметим, что можно избежать дублирования двух строк, «свернув» их во внутренний цикл, но это вряд ли сильно упростит запись:

```
for k:=1 to 2 do begin
 p := Pos(' ', s);
```

```
  s := Copy(s, p+1, Length(s)-p);
 end;
```

11) в пп. 8-10 описан достаточно общий метод, при котором инициалы могут быть любой длины, (но без пробела); в данном случае в условии четко сказано, что инициалы представляют собой именно 4 символа (буква, точка, буква, точка), поэтому можно найти первый пробел, а затем взять «хвост», который идет через 6 символов от него:

p := Pos(' ', s);	или	**p := Pos(' ', s);**
s := Copy(s,p+6,Length(s));	так	**Delete(s, 1, p+5);**

12) для преобразования номера школы из символьного вида в числовой можно использовать функцию **Val**:

Val(s, k, r);

эта процедура (*Turbo Pascal, Borland Pascal, PascalABC,* среда *АЛГО*) преобразует символьную строку **s** в числовое значение **k**; с помощью переменной **r** обнаруживается ошибка: если раскодировать число не удалось (в строке не число), в **r** будет записан нуль (здесь мы не будем обрабатывать эту ошибку, полагая, что все данные правильные);

если вы работаете на *ПаскалеABC* (никто не может вам запретить написать, что этот так), вместо **Val** можно использовать более удобную и понятную функцию **StrToInt**:

k := StrToInt(s);

13) таким образом, основной цикл выглядит так:

```
for i:=1 to N do begin
 readln(s); { читаем очередную строку }
  { выделяем часть после второго пробела }
 p := Pos(' ', s);
  Delete(s, 1, p+5);
  { определяем номер школы k }
  Val(s, k, r);
  C[k] := C[k] + 1; { увеличиваем счетчик k-ой школы }
end;
```

14) дальше стандартным алгоритмом определяем в массиве **C** минимальный элемент **Min**, не учитывая нули (школы, из которых не было участников):

```
Min := N;
for k:=1 to 99 do
 if (C[k] <> 0) and (C[k]<Min) then Min := C[k];
```

здесь интересна первая строчка, **Min:=N**: по условию всего было **N** участников, поэтому минимальное значение не может быть больше **N**; обратите внимание, что привычный вариант (который начинается с **Min:=C[1]**) работает неверно, если из первой школы не было ни одного участника

15) и выводим на экран номера всех школ (обратите внимание – **номера!**), для которых **C[k]=Min**:

```
for k:=1 to 99 do
 if C[k] = Min then writeln(k);
```

16) остается «собрать» программу, чтобы получилось полное решение; максимальное количество школ мы задали в виде константы **LIM**:

```
const LIM = 99;
var C:array[1..LIM] of integer;
  i, p, N, k, r, Min: integer;
  s:string;
```

```
begin
for k:=1 to 99 do C[k]:=0;
readln(N);
for i:=1 to N do begin
 readln(s); { читаем очередную строку }
  { выделяем часть после второго пробела }
 p := Pos(' ', s);
 Delete(s, 1, p+5);
  { определяем номер школы k }
 Val(s, k, r);
 C[k] := C[k] + 1; { увеличиваем счетчик k-ой школы }
end;
Min := N;
for k:=1 to LIM do
 if (C[k] <> 0) and (C[k]<Min) then Min := C[k];
for k:=1 to LIM do
 if C[k] = Min then writeln(k);
end.
```

На что обратить внимание:

- внимательно читайте условие, убедитесь, что вы понимаете смысл каждой строчки; для каждой мелочи постарайтесь определить, зачем она добавлена в условие, что она дает для решения задачи, что ограничивает, что не разрешает делать
- определите, какая именно информация из условия нужна для решения задачи, а какая – не нужна
- определите, что именно требуется вывести на экран в результате работы программы
- начинайте составлять программу с больших блоков, записывая ее сначала на псевдокоде, а потом уточняя детали
- проверяйте «крайние» варианты (например, возможность выхода за границы массива)
- проверьте, правильно ли заданы (и заданы ли вообще) начальные значения для всех переменных
- будьте внимательны, когда в массиве есть «мертвые» элементы, которые не нужно учитывать; проверяйте, что в этом случае ваши алгоритмы (например, поиск минимального элемента) работают правильно
- проверьте, правильно ли расставлены операторные скобки **`begin-end`**, ограничивающие тело цикла; их обязательно нужно ставить, если в теле цикла несколько операторов
- при использовании функции **`Pos`** не забывайте, что первый параметр – **что** ищем (образец), а второй – **где** ищем
- чтобы эксперту было легче понять вашу программу (особенно, если она получилась «нестандартной»), пишите комментарии; объясняйте, что хранится в основных переменных
- если это возможно, желательно работать только с целыми числами; этим вы избежите проблем, связанных с округлением и неточностью хранения дробных вещественных чисел в памяти компьютера

Задача 27.2

На вход программе подаются сведения о сдаче экзаменов учениками 9-х классов некоторой средней школы. В первой строке сообщается количество учеников N, которое не меньше 10, но не превосходит 100, каждая из следующих N строк имеет следующий формат:

<Фамилия> <Имя> <оценки>,

где <Фамилия> – строка, состоящая не более чем из 20 символов, <Имя> – строка, состоящая не более чем из 15 символов, <оценки> – через пробел три целых числа, соответствующие оценкам по пятибалльной системе. <Фамилия> и <Имя>, а также <Имя> и <оценки> разделены одним пробелом. Пример входной строки:

Иванов Петр 4 5 3

Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет выводить на экран фамилии и имена трех худших по среднему баллу учеников. Если среди остальных есть ученики, набравшие тот же средний балл, что и один из трех худших, то следует вывести и их фамилии и имена.

Как правильно понимать условие?

1) как и в предыдущей задаче, данные «подаются на вход программе», то есть, их можно читать с помощью операторов **read** (**readln**), предполагая, что кто-то вводит эти данные с клавиатуры вручную
2) «количество учеников не меньше 10, но не превосходит 100», здесь только вторая часть – полезная информация, она намекает на то, что придется все введенные данные одновременно держать в памяти, выделив массив (или массивы) размером 100 элементов
3) сказано, что фамилия имеет длину не более 20 символов, а имя – не более 15; здесь, по сути, важно лишь то, что фамилия и имя (вместе) занимают меньше 255 символов, то есть, «влезут» в стандартное ограничение (255 символов) для типа **string** в классических версиях Паскаля
4) после фамилии и имени записаны три оценки (а не одно число, как в прошлой задаче), причем по условию нас НЕ интересуют эти числа, а интересует только средний балл каждого ученика;
5) важно! средний балл – это вещественное число (может иметь дробную часть), тут уже стоит задуматься: все задачи обычно составляются так, чтобы они решались «хорошо», в то же время операции с дробными числами (почти) всегда выполняются с ошибками, поскольку большинство вещественных чисел нельзя точно (стандартными методами) представить в памяти реального компьютера
6) следующий шаг к правильному решению: поскольку число оценок у всех учеников одинаковое, средний балл для каждого это сумма его оценок, деленная на 3; поэтому вместо среднего балла мы можем сравнивать суммы баллов – целые числа!
7) требуется вывести фамилии и имена (баллы не нужны!) трех худших учеников, причем их может быть и больше, если несколько «худших» набрали одинаковую сумму баллов
8) если бы требовался один худший – все решается поиском по массиву; первая идея – найти самого худшего (1 проход), затем – 2-ого с конца (еще 1 проход), и, наконец, 3-его (всего три прохода по массиву)
9) это не лучший вариант (на экзамене будут сняты баллы) по двум причинам:
   - в таком методе решения три прохода по массиву, а в самом деле достаточно одного (см. далее), значит, программа неэффективна

- непонятно, что делать в том случае, если худших – больше трех (в предельном случае – вообще все!) – за это также снимут баллы (программа работает не для всех вариантой входных данных)

10) возникает следующий вариант – отсортировать массив про возрастанию суммы (и, следовательно, среднего балла), одновременно переставляя имена и фамилии, а затем вывести самых худших, которые после сортировки окажутся в начале массива
11) этот вариант тоже плох, потому что программа неэффективна; «школьные» алгоритмы сортировки (метод «пузырька», метод выбора) имеют сложность $O(N^2)$, а надо попытаться найти метод со сложностью $O(N)$

Решение (общий подход):

Сначала составим программу в самом общем виде на псевдокоде, чтобы определить ее основные блоки, а потом будем их постепенно «расшифровывать» через операторы языка программирования:

{ читаем все данные и запоминаем их }
{ находим три худших результата }
{ выводим фамилии и имена тех, чей результат меньше или
равен «третьему худшему» }

До того, как начать писать «нормальный» код, нужно определить, как хранить данные; в данном случае нужно запомнить несколько данных по каждому ученику, их удобнее объединить в запись с двумя полями (фамилия-имя и сумма баллов); таких записей нужно выделить в памяти не менее 100 (по условию), то есть, массив из 100 элементов:

```
Const lim=100;
var Info: array[1..LIM] of record
       name: string;
       sum: integer;
     end;
```

Чтение данных:

после того, как мы прочитали фактическое число учеников N, в цикле считываем и расшифровываем информацию о них, сохраняя все данные в структурах

```
for i:=1 to N do begin
 { считываем строку данных }
 Info[i].name := { фамилия и имя };
 Info[i].sum := { сумма баллов };
end;
```

здесь, в принципе, можно использовать тот же подход, что и в первой задаче – читаем строку целиком, затем «разбираем» ее на части с помощью стандартных функций – однако, для разнообразия, мы используем другой подход – будем читать информацию посимвольно, то есть, считывая по одному символу в переменную с типа char;

сначала в поле name очередной структуры записываем пустую строку "(в которой нет ни одного символа, длина равна нулю)

```
Info[i].name := ''; { пустая строка }
```

затем считываем символы фамилии и сразу приписываем их в конец поля

```
name:
repeat
 read ( c );
 Info[i].name := Info[i].name + c;
until c = ' '; { пока не прочитали пробел }
затем также читаем из входного потока имя, до пробела, и записываем
его в конец того же поля name:
repeat
 read ( c );
```

```
 Info[i].name := Info[i].name + c;
until c = ' '; { пока не прочитали пробел }
```

заметьте, что эти два цикла одинаковы, поэтому ввод имени и фамилии можно записать в виде вложенного цикла так:

```
Info[i].name := ''; { пустая строка }
for k:=1 to 2 do
 repeat
  read ( c );
  Info[i].name := Info[i].name + c;
 until c = ' '; { пока не прочитали пробел }
```

важно! обратите внимание, что для организации внутреннего цикла используется другая переменная, k (а не i, потому что i – переменная главного цикла, она обозначает номер текущего ученика)

теперь во входном потоке остались три числа, которые мы можем последовательно считывать в целую переменную mark, а затем – добавлять к полю `Info[i].sum`:

```
 Info[i].sum := 0;
 for k:=1 to 3 do begin
  read(mark);
  Info[i].sum := Info[i].sum + mark;
 end;
 readln;
```

последняя команда readln пропускает все оставшиеся символы до новой строки (из этой мы прочитали все, что нужно)

вот полный цикл ввода данных, после его окончания все исходные данные будут записаны в первые N записей массива Info:

```
for i:=1 to N do begin
  { ввод имени и фамилии }
 Info[i].name := '';
 for k:=1 to 2 do
  repeat
   read(c);
   Info[i].name := Info[i].name + c;
  until c = ' ';
  { ввод и суммирование оценок }
 Info[i].sum := 0;
 for k:=1 to 3 do begin
  read(mark);
  Info[i].sum := Info[i].sum + mark;
 end;
 readln;
end;
```

Поиск трех худших данных:

1) теперь нужно придумать, как за один проход по массиву найти три худших результата;
2) как бы мы решили эту задачу, если бы нам нужно было просмотреть столбик чисел и найти три минимальных? можно сделать, например, так:
   - на бумажке вести записи в три столбика, в первом записывать минимальное число, в втором – следующее по величине, в третьем – «третье минимальное»
   - сначала пишем первое число в первый столбик, оно – минимальное, потому что других мы не еще видели; пусть это число 14:

минимум	второе	третье

14		

- пусть следующее число – 12; оно меньше минимального, поэтому его нужно записывать в первый столбец, а «старое» минимальное число «переедет» во второй столбец

минимум	второе	третье
~~14~~		
12	14	

- пусть дальше идет число 10 – теперь оно станет минимальным, его нужно записывать в первый столбец; при этом 12 «переедет» из первого столбца во второй, а 14 – из второго в третий

минимум	второе	третье
~~14~~		
~~12~~	~~14~~	
10	12	14

- пусть следующее число – 11; оно больше минимального, но меньше «второго», поэтому его нужно поставить во второй столбец; число 12 из второго столбца перемещается в третий, а число 14 из третьего столбца удаляется из кандидатов в «три минимальных»

минимум	второе	третье
~~14~~		
~~12~~	~~14~~	
10	~~12~~	~~14~~
	11	12

- просмотрев таким образом весь столбик чисел, за один проход (!) можно найти три минимальных элемента
- остается только переложить этот алгоритм на язык программирования

3) выделим в памяти три целых переменных: **min1** (минимальный), **min2** («второй минимальный»), **min3** («третий минимальный»), в виде начальных значений запишем в каждую из них число, заведомо превышающее максимальную возможную сумму трех оценок, например, 20 (**>5+5+5**)
4) полный цикл поиска выглядит так:

```
min1 := 20; min2 := 20; min3 := 20;
for i:=1 to N do begin
 if Info[i].sum < min1 then begin { новый min1 }
  min3 := min2; min2 := min1;
  min1 := Info[i].sum;
 end
 else if Info[i].sum < min2 then begin { новый min2 }
   min3 := min2;
   min2 := Info[i].sum;
 end
  else if Info[i].sum < min3 then { новый min3 }
     min3 := Info[i].sum;
end;
```

5) обратим внимание на два момента: во-первых, когда переезжают два элемента, сначала нужно перемещать второй на место третьего, а потом – первый на место второго:

```
min3 := min2;
min2 := min1;
```

эти операторы нельзя менять местами, иначе «старое» значение **min2** будет потеряно;

во-вторых, если проверять условие **Info[i].sum < min2** нужно только тогда, когда очередная сумма не меньше, чем **min1**, поэтому каждый следующий условный оператор стоит в **else**-блоке предыдущего, то есть, выполняется только тогда, когда предыдущий не сработал

6) итак, мы нашли три минимальных результата, и остается вывести на экран фамилии и имена тех, у кого сумма баллов меньше или равна **min3**:

```
for i:=1 to N do
 if Info[i].sum <= min3 then
   writeln(Info[i].name);
```

7) на всякий случай приведем полную программу, она получилась довольно длинная

```
const LIM = 100;
var Info: array[1..LIM] of record
      name: string;
      sum: integer;
     end;
 i, k, N, mark, min1, min2, min3: integer;
  c: char;
begin
readln(N);
  { ввод исходных данных }
for i:=1 to N do begin
 Info[i].name := '';
 for k:=1 to 2 do
  repeat
   read(c);
   Info[i].name := Info[i].name + c;
  until c = ' ';
 Info[i].sum := 0;
 for k:=1 to 3 do begin
  read(mark);
  Info[i].sum := Info[i].sum + mark;
 end;
 readln;
end;
  { поиск трех минимальных }
min1 := 20; min2 := 20; min3 := 20;
for i:=1 to N do begin
 if Info[i].sum <min1 then begin
  min3 := min2; min2 := min1;
  min1 := Info[i].sum;
 end
 else if Info[i].sum <min2 then begin
   min3 := min2;
   min2 := Info[i].sum;
  end
```

```
   else if Info[i].sum <min3 then
      min3 := Info[i].sum;
end;
  { вывод результата }
for i:=1 to N do
 if Info[i].sum <= min3 then
   writeln(Info[i].name);
end.
```

8) эту задачу можно решить и без записей, используя два массива: массив символьных строк **name** и массив целых чисел **sum**, они объявляются так:

```
var name: array[1..MAX] of string;
  sum: array[1..MAX] of integer;
```

после этого в приведенной программе нужно заменить везде **Info[i].name** на **name** и **Info[i].sum** на **sum.**

На что обратить внимание:

1. в исходных данных выделите то, что не нужно для решения задачи; при чтении эти части можно просто пропускать;
2. если нам не нужны фамилия и имя отдельно, можно хранить их вместе, в виде одной строки
3. если нас интересует только сумма оценок, не нужно хранить их в памяти по отдельности
4. если можно при решении задачи обойтись без вещественных чисел, сделав все вычисления только с целыми числами – нужно поступить именно так (иначе снимут баллы), поскольку операции с вещественными числами во многих случаях случаев выполняются неточно
5. алгоритм сложности $O(N^2)$ (например, сортировку) нужно использовать только тогда, когда нет алгоритма сложности $O(N)$; как правило, в задачах ЕГЭ такой алгоритм всегда можно (попытаться) найти; за неэффективный алгоритм при оценке решения будут сняты баллы

За что снимают баллы:

- программа работает не для всех исходных данных, не обрабатывает некоторые частные случаи
- неверно реализован алгоритм поиска минимального элемента, сортировки и т.п.
- неэффективность алгоритма:
  - используется алгоритм, имеющий сложность $O(N^2)$, когда есть алгоритм сложности $O(N)$
  - используется несколько проходов по массиву, когда достаточно одного
  - лишний расход памяти (используются дополнительные массивы или размер массива определен неверно)
  - используются операции с вещественными числами, когда можно все решить в целых числах
- переменная не описана или описана неверно
- переменным не присвоены нужные начальные значения (например, не обнуляются счетчики) или присвоены неверные значения
- нет вывода результата в конце программы

- перепутаны знаки < и >, логические операции **or** и **and**
- применяется недопустимая операция, например, **div** или **mod** для вещественных чисел
- неверно расставлены операторные скобки **begin-end**
- в цикле **for** используется вещественная переменная (Паскаль)
- в цикле **while** или **repeat** не изменяется переменная цикла, из-за чего происходит зацикливание
- синтаксические ошибки (знаки пунктуации – запятые, точки, точки с запятой; неверное написание ключевых слов); чтобы получить 4 балла, при абсолютно верном решении нужно сделать не более одной синтаксической ошибки; на 3 балла – до трех ошибок, на 2 балла – до пяти и на 1 балл – до семи ошибок

Типичные ошибки учащихся, допущенные при выполнении задания 27:

- вследствие неверного понимания условия, решается другая задача;
- некорректно считываются входные данные;
- неверно строчные буквы преобразуются в прописные;
- при обработке входных данных не проверяется, является ли символ английской буквой;
- входные данные сохраняются в массиве символов или строке;
- программа реализует неэффективный алгоритм (например, используется сортировка строки или массива символов);
- выводится не последнее слово в алфавитном порядке или слово не максимальной длины;
- программа содержит алгоритмические ошибки: выход за границу массива, используется знак “<” вместо “<=”, “/” вместо “\”, “or” вместо “and”, div вместо mod и т.п.;
- большое количество синтаксических ошибок (пропущен или неверно указан знак пунктуации, неверно написано или пропущено зарезервированное слово языка программирования, не описана или неверно описана переменная, применяется операция, недопустимая для соответствующего типа данных).

Выполнение заданий 2 части требует высокого уровня подготовки учащихся в области программирования, достижение которого невозможно при обучении информатике на базовом уровне. Необходимо организовать дополнительное обучение учащихся в рамках, например, элективных курсов, участвовать в олимпиадах по программированию.

Для успешной сдачи ЕГЭ по информатике требуется специальная подготовка, знакомство и разбор заданий демонстрационных вариантов КИМ, заданий открытого сегмента ФИПИ (www.fipi.ru), знакомство с критериями оценивания заданий 2 части.

При подготовке к ЕГЭ следует обратить внимание учащихся на то, что ответы на задания третьей части работы должны быть записаны четко, понятным почерком, в строгом соответствии с требованиями.

АПРОБАЦИЯ.

Итоги работы по подготовке выпускников по описанной методике таковы:

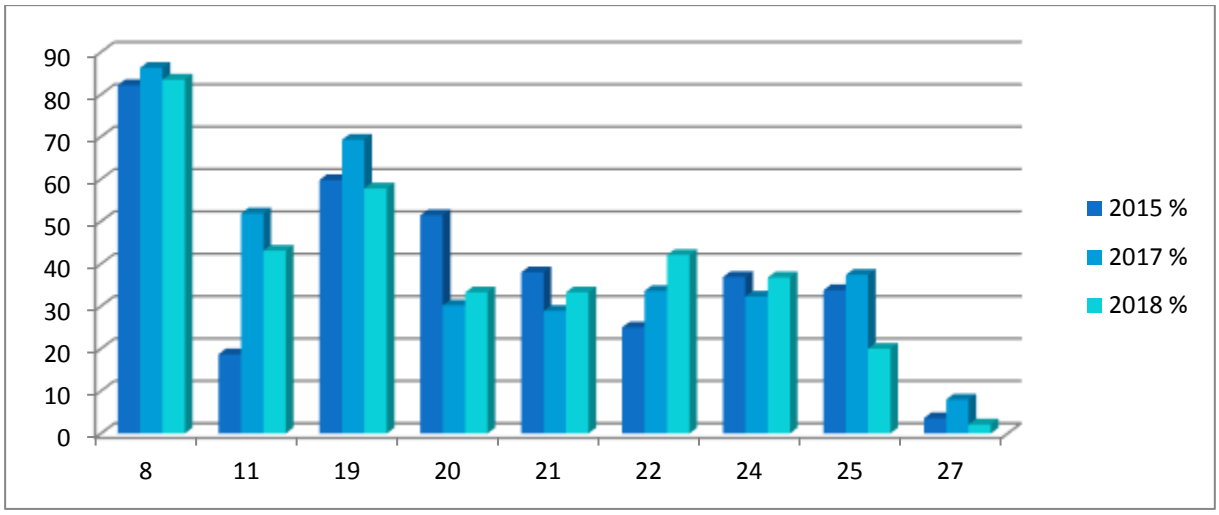
Результаты ЕГЭ в МОУ Пыщугская СОШ (высокобальники)

№ п/п	Проверяемые элементы содержания и виды деятельности	Код участника 2015	Код участника 2017	Код участника 2017	% справившихся с заданием
		1406-8755-5535	0409-8023-8556	1487-3359-6076	
8	Знание основных конструкций языка программирования, понятия переменной, оператора присваивания	+	+	+	100
11	Умение исполнить рекурсивный алгоритм	+	+	+	100
19	Работа с массивами (заполнение, считывание, поиск, сортировка, и др.)	+	+	+	100
20	Анализ программ с циклами и условными операторами Анализ алгоритма, содержащего вспомогательные алгоритмы, цикл и ветвление	+	+	+	100
21	Умение анализировать программу, использующую процедуры и функции	+	-	+	66,6
22	Умение анализировать результат исполнения алгоритма	+	+	+	100
24	Умение прочесть фрагмент программы на языке программирования и исправить допущенные ошибки	3(3)	3(3)	3(3)	100
25	Умения написать короткую (10-15 строк) простую программу (например, обработки массива) на языке программирования	0(2)	2(2)	2(2)	66,6
26	Дерево игры. Поиск выигрышной стратегии. (Динамическое программирование)	3(3)	3(3)	3(3)	100
27	Умения создавать собственные программы (30-50 строк) для решения задач средней сложности	2(4)	1(4)	0(4)	48
% верных		85	82	82	-
Балл на ЕГЭ		84	83	83	-

Конечно, не все наши выпускники показывают такие результаты. Низкие результаты показали выпускники 2018 года. (Сдали все, но показали низкий балл -50). Причиной была их недостаточно высокая мотивация, экзамен выбирался слабыми учениками, не участвующими в олимпиадах по программированию. Хотя двое из них поступили в университеты на информационые технологии. В этот год экзамен выбрали 5 человек, 1 девушка и 4 юноши. Двое из них готовятся очень серьёзно, будем надеятся на хорошие результаты ЕГЭ у них.

Результаты по заданиям программирования ЕГЭ Костромская область

№ п/п	Проверяемые элементы содержания и виды деятельности	2015 %	2017 %	2018 %	Средний % справившихся с заданием
8	Знание основных конструкций языка программирования, понятия переменной, оператора присваивания	82,2	86,3	83,5	84
11	Умение исполнить рекурсивный алгоритм	18,7	51,9	43,1	37,9
19	Работа с массивами (заполнение, считывание, поиск, сортировка, и др.)	59,8	69,3	57,8	62,3
20	Анализ программ с циклами и условными операторами Анализ алгоритма, содержащего вспомогательные алгоритмы, цикл и ветвление	51,5	30,2	33,3	38,3
21	Умение анализировать программу, использующую процедуры и функции	38	28,9	33,3	33,4
22	Умение анализировать результат исполнения алгоритма	25	33,6	42,1	33,56
24	Умение прочесть фрагмент программы на языке программирования и исправить допущенные ошибки	36,9	32,3	36,8	35,33
25	Умения написать короткую (10-15 строк) простую программу (например, обработки массива) на языке программирования	33,8	37,4	20	30,4
27	Умения создавать собственные программы (30-50 строк) для решения задач средней сложности	3,6	1,2	2,1	2,3
Средний балл ЕГЭ		56,31	58,06	58,07	-



Первые две задачи второй части (задания 24 и 25) представляют раздел программирования (умение написать программу и провести анализ готовой программы) показывают невысокие показатели по результатам ЕГЭ в регионе, всё-таки наблюдается повышение количества решений задачи24.

Вызывает тревогу очень низкий показатель выполнения задания 27 на самостоятельное программирование". Согласно публикуемым планам вариантов КИМ (спецификация контрольных измерительных материалов) эта задача на умение создавать собственные программы (30–50 строк) для решения задач средней сложности. За последние года три–четыре наблюдается повышение уровня сложности этой задачи: она всё больше стремится к задачам олимпиадного уровня. Учитывая малое количество часов, выделяемых на предмет в школьном курсе, необходимо думать над решением этой ситуации.

ЗАКЛЮЧЕНИЕ
ВЫВОДЫ

Знания технологий программирования означают, что каждый год тысячи выпускников, успешно сдавших ЕГЭ по информатике, являются начинающими профессионалами в программировании. Такого нет и не было ни в одной стране мира. не случайно наши российские студенты уже десять лет являются чемпионами мира по программированию.

Это – конкурентное преимущество страны

В нашей школе выпускники 2015-2018 годов, успешно сдавших ЕГЭ по информатике учатся в МГТУ им. Баумана на факультетах Робототехника и комплексная автоматизация, Аэрокосмический, на факультетах информационных технологий в Котромском и Нижегородском университетах.

Но программирование нужно не только «избранным». На уроках программирования дети учатся в первую очередь работать с информацией, структурировать её, управлять ею, а эти навыки жизненно необходимы в условиях все нарастающего «информационного вала» современной жизни. Даже приблизительное понимание, как устроен компьютер, как он работает и исполняет программы, каковы его возможности и ограничения, — важный навык в нынешних условиях, когда компьютеры проникли буквально повсюду. Даже если ребенок и не станет программистом, приобретенные во время занятия программированием навыки будут для него хорошим подспорьем в будущей жизни.

Поскольку контроль является неотъемлемой частью учебного процесса, то всё происходящее в организации государственного итогового контроля не может не отразиться на организации учебного процесса и промежуточном контроле знаний учащихся. Подготовка учащихся к ЕГЭ – это длительная и кропотливая работа учителя, в результате которой необходимо обращать внимание на следующие моменты:

1)«Учить только хорошему». Ученики должны сразу, с первого занятия видеть перед собой правильные, хорошие цели и правильные, хорошие примеры. Если не обратить внимания на какие-то вещи (например, форматирование кода), пустить их на самотек, дети сделают это так, как «поймут» сами. Впоследствии их придется переучивать, а это всегда намного менее продуктивно, чем учить правильно с самого начала. Поэтому на первом же занятии учащиеся узнают, как правильно пользоваться пробелами и отступами, и почему важны пустые строки, разбивающие программу на логические фрагменты. С первых же занятий вводится понятие качества имен, и от детей требуется использовать понятные имена для переменных и функций.

«Учить программированию, а не языку». Все понятия, даваемые ученикам, выводятся как инструмент решения проблемы. Даже не совсем так: сначала формулируется проблема, дается возможность её «пощупать», попробовать решить имеющимися средствами (в качестве домашнего задания или вместе с учителем в классе). Например, мы умеем рисовать на экране домик (у нас уже есть такая функция). Давайте нарисуем на экране 5 домиков друг за другом. Задачу, безусловно, можно решить, вызвав функцию 5 раз. Но, проверяя технологию, мы задаем вопросы: а если надо будет 10 домиков? 50? 100? А если 4? А если столько, сколько поместится на экране? Или столько, сколько введет пользователь программы? Затем детям предлагается обсудить, как можно было бы решить эту проблему, и обычно они сами, с некоторой помощью преподавателя, формулируют с той или иной степенью приближения идею цикла. Лишь только затем рассказывается синтаксис оператора цикла в выбранном для обучения языке программирования.

Искать новые методики обучению программированию. Например, методика Ильи Рудольфовича Дединского.

2) Тестирование как новая форма экзамена накапливает свой опыт и требует предварительной подготовки. В связи с этим учителям следует активнее вводить тестовые технологии в систему обучения уже с 7 класса. Тренировки в выполнении тестовых заданий позволят реально повысить тестовый балл. Зная типовые конструкции тестовых заданий, ученик практически не будет тратить время на понимание инструкции. Кроме того, во время таких тренировок формируются соответствующие навыки психологической саморегуляции и самоконтроля, позволяющие мобилизовать себя в решающей ситуации, овладеть собственными эмоциями, способствуют развитию навыков мыслительной работы.

3) Тестовая работа должна быть выполнена в строго отведенное время. Поэтому нужно учить обучающихся правильно ориентироваться во времени. Для этой цели могут проводиться диагностические замеры – небольшие проверочные работы, требующие выполнения заданий в уме и фиксирование только окончательного ответа, причём в строго отведённое время.

4) Особое внимание следует уделять работе с формулировками, характерными для экзаменационных материалов. Часто непривычная формулировка сбивает с толку даже вполне подготовленного ученика. Важной составляющей работы является сведение к минимуму эффекта неожиданности. Подбирая тренировочные задачи, нужно предлагать возможно большее число вариантов формулировок. Ученик постепенно привыкает к этому разнообразию, учиться вдумчиво читать условие, искать неявные смыслы в тексте.

5) На уроках информатики в старшем звене основное внимание должно быть направлено на овладение умениями извлекать информацию из условия и требования задачи, вычленять отдельные элементы, комбинировать их, выводить следствия, переформулировать требования задачи. Поэтому один из элементов работы учителя – это учить учащихся анализу условия решаемой задачи. Кроме того, необходимо в обязательном порядке проводить анализ заданий после проведения тестовых работ.

6) Важное место следует отводить организации повторения изученного материала, особенно организации заключительного повторения. В процессе повторения память у учащихся развивается. Повторение учебного материала необходимо осуществлять во всей системе изученного процесса:

- При изложении новых понятий
- При закреплении изученного ранее
- При организации самостоятельных работ разных видов
- При организации обобщающего повторения

7) Организовывая процесс повторения учебного материала необходимо уделять значительное внимание таким дидактическим приёмам как сравнение, синтез, анализ, обобщение, классификация, которые способствуют активному протеканию процесса запоминания. Для осознанного восприятия материала необходимо привлекать учащихся к такому виду работы, как составление упражнений по образцу.

8) При организации итогового повторения должен быть отобран самый важный материал. Целесообразно весь повторяемый материал распределить по методическим линиям курса

9) Нужно работать над мотивацией обучающихся к участию в итоговой аттестации. Ученик должен иметь определенную цель, которая поможет ему в сдаче экзамена.

10) Поддерживать тесную связь с родителями, проводить для родителей анализ пробных ЕГЭ и вообще родители должны быть в курсе состояния уровня подготовки их детей к итоговой аттестации.

11) Осуществлять межпредметную связь, особенно с такими предметами, как математика.

12) Осуществлять на уроках индивидуально-дифференцированный подход, применять современные образовательные технологии..

В заключение, хочется поблагодарить К.Ю Полякова за предоставленные материалы по подготовке к ЕГЭ и разностороннюю поддержку моей методической разработки.

Список литературы

1. http://kpolyakov.spb.ru/school/ege.htm
2. http://school-collection.edu.ru
3. https://inf-ege.sdamgia.ru/
4. Абрамян М.Э., Михалкович С.С., Русанова Я.М., Чердынцева М.И. Информатика. ЕГЭ шаг за шагом. — М.: НИИ школьных технологий, 2010.
5. Босова Л.Л., Босова А.Ю. Информатика: Учебник для 10 класса. – М.: БИНОМ. Лаборатория знаний, 2017.
6. Босова Л.Л., Босова А.Ю. Информатика: Учебник для 11 класса. – М.: БИНОМ. Лаборатория знаний, 2017
7. Босова Л.Л., Босова А.Ю. Электронное приложение к учебнику «Информатика. 10 -11 класс»
8. Волкова И. Б., Штепа Ю. П. Изучение алгоритмической конструкции «Цикл» в базовом курсе информатики // Современная педагогика. – 2014. – № 12 (25). – С. 4-11.
9. Гусева И.Ю. ЕГЭ. Информатика: раздаточный материал тренировочных тестов. — СПб: Тригон, 2014.
10. Демонстрационные варианты ЕГЭ 2004-2018 гг.
11. Зорина Е.М., ЕГЭ 2019. Информатика. Сборник заданий: 350 заданий с ответами. — М.: Эксмо, 2018.
12. Козлов С. В. Анализ выполнения тестовых заданий части 1 ЕГЭ (ОГЭ) по информатике и ИКТ в 2014 году в контексте организации профильного обучения // Современная педагогика. – 2014. – № 10 (23). – С. 56-64.
13. Козлов С. В. Анализ выполнения тестовых заданий части 2 ЕГЭ (ОГЭ) по информатике и ИКТ в 2014 году в контексте организации профильного обучения // Современная педагогика. – 2014. –№ 12 (25). – С. 111-123.
14. Козлов С. В. Анализ выполнения тестовых заданий части 3 ЕГЭ (ОГЭ) по информатике и ИКТ в 2014 году в контексте организации профильного обучения // Современная педагогика. – 2015. – № 1 (26). – С. 96-106.
15. Козлов С. В. Вопросы формирования индивидуального теста // Гуманитарные научные исследования. – 2014. – № 10 (38). – С. 64-70.
16. Козлов С. В. Методические рекомендации использования автоматизированной дидактической системы индивидуального тестирования // Психология, социология и педагогика. – 2014. – № 10 (37). – С. 22-26.
17. Козлов С. В. О подготовке школьников к участию в олимпиадах по информатике // Психология, социология и педагогика. – 2015. – № 1 (40). – С. 68-74.
18. Козлов С. В. Обобщенный анализ выполнения тестовых заданий ЕГЭ (ОГЭ) по информатике и ИКТ в 2014 году // Современная педагогика. – 2015. – № 6 (31). – С. 9-16.
19. Козлов С. В. Особенности обучения школьников информатике в профильной школе // Научно-методический электронный журнал «Концепт». – 2014. – № 1. – С. 31-35. ART 14006. – URL: http://e-koncept.ru/2014/14006.htm.
20. Козлов С. В. Педагогическое проектирование индивидуального тестирования в личностно ориентированной обучающей системе: дис. … канд. пед. наук: 13.00.01 и 13.00.02: защищена 24.05.06: утв. 20.11.06 / Козлов Сергей Валерьевич. – Смоленск, 2006. – 204 с.
21. Козлов С. В. Педагогическое проектирование индивидуального тестирования в личностно ориентированной обучающей системе: автореферат дис. … канд. пед. наук. – Смоленск, 2006. – 18 с.

22. Козлов С. В. Применение методов функционального анализа при формировании оптимальных стратегий обучения школьников // Международный журнал экспериментального образования. – 2016. – № 3-2. – С. 182-185; URL: http://www.expeducation.ru/ru/article/view?id=9696 (дата обращения: 21.04.2016).
23. Крылов С.С., Лещинер В.Р., Якушкин П.А. ЕГЭ-2017. Информатика. Универсальные материалы для подготовки учащихся / под ред. В.Р. Лещинера / ФИПИ. — М.: Интеллект-центр, 2017.
24. Крылов С.С., Ушаков Д.М. ЕГЭ 2010. Информатика. Тематическая рабочая тетрадь. — М.: Экзамен, 2010.
25. Крылов С.С., Ушаков Д.М. ЕГЭ 2018. Информатика. Тематические тестовые задания. — М.: Экзамен, 2018.
26. Крылов С.С. ЕГЭ 2019. Тренажёр. Информатика. — М.: Экзамен, 2018.
27. Л.Н. Евич Информатика и ИКТ Подготовка к ЕГЭ 2018//Легион/Ростов на Дону/ 2018
28. Максимова Н. А. Моделирование образовательной среды личностного развития учащихся // Бюллетень науки и практики. – 2016. – № 5 (6). – С. 481-484.
29. Максимова Н. А. Развитие логического мышления учащихся с использованием информационных технологий // Современные проблемы науки и образования. – 2014. – № 5. – С. 32.
30. Материалы авторской мастерской Босовой Л.Л. (metodist.lbz.ru/)
31. Морозова Е. В., Максимова Н. А. Применение аудиовизуальных технологий обучения в системе развития логического мышления учащихся // Научно-методический электронный журнал Концепт. – 2015. – Т. 13. – С. 466-470.
32. Поляков К.Ю., Е.А. Еремин Информатика. 10 класс. Углубленный уровень: учебник в 2 ч. – БИНОМ. Лаборатория знаний, 2017 г.
33. Поляков К.Ю., Е.А. Еремин Информатика. 11 класс. Углубленный уровень: учебник в 2 ч. – БИНОМ. Лаборатория знаний, 2017 г.
34. Салиновская Е. В., Штепа Ю. П. Методические аспекты изучения процесса передачи информации в школьном курсе информатики // Психология, социология и педагогика. – 2014. – № 11 (38). – С. 58-62.
35. Самылкина Н.Н., Синицкая И.В., Соболева В.В., ЕГЭ 2019. Информатика. Задания, ответы, комментарии. — М.: Эксмо, 2018.
36. Самылкина Н.Н., Синицкая И.В., Соболева В.В., ЕГЭ 2019. Тематические тренировочные задания. — М.: Эксмо, 2018.
37. Семакин И.Г., Е.К. Хеннер, Т.Ю. Шеина Информатика. 10 класс. Базовый уровень: учебник – БИНОМ. Лаборатория знаний, 2017 г.
38. Скокова О. В., Штепа Ю. П. Методические особенности формирования представлений учащихся об операционной системе // Гуманитарные научные исследования. – 2014. – № 12-1 (40). – С. 97-105.
39. Тишин В. И. Программирование на Паскале. Практикум М.: БИНОМ. Лаборатория знаний, 2017.
40. Тренировочные работы МИОО и СтатГрад.
41. Ушаков Д.М. ЕГЭ-2017. Информатика. 20 типовых вариантов экзаменационных работ для подготовки к ЕГЭ. — М.: Астрель, 2017.
42. Чуркина Т.Е. ЕГЭ 2017. Информатика. Тематические тренировочные задания. — М.: Эксмо, 2017.
43. Якушкин П.А., Лещинер В.Р., Кириенко Д.П. ЕГЭ 2010. Информатика. Типовые тестовые задания. — М.: Экзамен, 2017.
44. Якушкин П.А., Ушаков Д.М. Самое полное издание типовых вариантов реальных заданий ЕГЭ 2018. Информатика. — М.: Астрель,2018

Список источников для подготовки к сдаче ЕГЭ по информатике:

1. Сайт Федерального института педагогических измерений http://www.fipi.ru/
2. Сайт Информатика - образовательный ресурс http://ege-go.ru/
3. EXAMER.RU(https://examer.ru/app/intro)
4. https://inf-oge.sdamgia.ru/ Сайт Дмитрия Гущина Решу ОГЭ https://vk.com/foxford_edu
5. Online тесты по информатике и информационным технологиям http://markx.narod.ru/inf/
6. Авторская мастерская Босовой Л.Л. http://metodist.lbz.ru/authors/informatika/3/umk5-9.php
7. Группы VK Фоксфорд Открытая группа
8. Контрольные задания Московского института открытого образования (МИОО) http://www.mioo.ru/ogl.php
9. Материалы сайта К. Полякова http://kpolyakov.narod.ru/
10. Образовательный ресурс http://infoegehelp.ru/
11. Обучающие программы по информатике http://markx.narod.ru/sch/
12. Подготовка к ЕГЭ и ЕГЭ по информатике-2008-2013 http://mymark.narod.ru/ege/
13. Портал КЛЯКС@NET http://www.klyaksa.net/
14. Сайт ЕГЭ с демоверсиями http://www.ege.edu.ru

ПРИЛОЖЕНИЯ

Приложение 1.Тесты для подготовки к ЕГЭ в 11 классе в программе My Test составитель Вагина Е.Н.

Приложение 2 Входная диагностическая работа для подготовки к ЕГЭ в 11 классе

Приложение 3 Загрузочный файл к программе My Test

Приложение 4. Тесты для подготовки к ЕГЭ в программе My Test

Приложение 5.Программы на языке PascalABC