ОТЛИЧНИК

ФИПИ
Федеральный институт педагогических измерений

ЕГЭ

ИНТЕЛЛЕКТ-ЦЕНТР

ИНФОРМАТИКА

РЕШЕНИЕ СЛОЖНЫХ ЗАДАЧ

97 100 98
95 99 96

ФЕДЕРАЛЬНЫЙ ИНСТИТУТ ПЕДАГОГИЧЕСКИХ ИЗМЕРЕНИЙ

ФИПИ

ОТЛИЧНИК ЕГЭ

ИНФОРМАТИКА

Решение сложных задач

«Интеллект-Центр»
2010

УДК 373.167.1:004
ББК 32.81я721
О-80

Авторы-составители:
Крылов С.С., Ушаков Д.М.

О-80 Отличник ЕГЭ. Информатика. Решение сложных задач / ФИПИ. – М.: Интеллект-Центр, 2010. – 152 с.

В данном пособии рассматриваются основные типы заданий с развернутым ответом части 3 (С), особенности их оценивания, а также приведены ответы выпускников на задания части С с выставленными баллами и комментариями разработчиков тестовых заданий. В книге разбираются типичные ошибки, допущенные выпускниками на экзамене. Пользователям пособия предложены различные типы задания части С для самостоятельного выполнения, которые использовались на экзаменах 2005–2009 гг. В конце пособия даны эталоны ответов с критериями оценивания.

Пособие адресовано в первую очередь учащимся и абитуриентам, которым предстоит сдавать ЕГЭ по информатике. Оно поможет подготовиться к выполнению наиболее трудной части экзаменационной работы. Книга, безусловно, будет интересна учителям школ, методистам и родителям.

Генеральный директор издательства «Интеллект-Центр»
М.Б. Миндюк

Редактор Д.П. Локтионов
Технический редактор В.С. Торгашова
Художественный редактор Е.Ю. Воробьева

Подписано в печать 23.11.2009 г. Формат 60x84/8.
Печать офсетная. Усл. печ. л. 19,0. Тираж 5000 экз.

ISBN 978-5-89790-620-8

ВСТУПЛЕНИЕ

Вы читаете эту книгу, чтобы подготовиться к сдаче экзамена по информатике в форме ЕГЭ. Экзамен – дело очень ответственное и, возможно, ваши успехи на нем повлияют на вашу дальнейшую судьбу. Нам бы хотелось помочь вам сдать этот экзамен наиболее эффективно, оказаться к нему наиболее подготовленными. Поэтому мы не станем обсуждать, хорошо или плохо составлены задания, а также хороша или плоха система оценивания выполнения этих заданий. Мы просто постараемся научить вас в полной мере проявить свои знания и способности.

ВАЖНЫЕ ЗАМЕЧАНИЯ

Прежде чем мы начнем обсуждать типичные задачи части С и их решения, мы должны напомнить вам, каким образом ваши решения будут проверяться.

Задания части С проверяют специально подготовленные люди – эксперты ЕГЭ. Они собираются в центре проверки, и каждому из них выдают толстую стопку экзаменационных работ. Эти работы предварительно отсканированы и напечатаны на специальных бланках, на которых нет никакой информации о том, чья это работа. Эксперты проверяют ваши работы до тех пор, пока не проверят все. Это тяжелый, не очень благодарный труд.

В ваших интересах соблюдать следующие простые правила:

1. Ответы и решения заданий части С нужно писать только в специально предназначенных для этого прямоугольных рамках. Ни в коем случае не заходите за границы этой рамки! Сканер просто не скопирует то, что выходит за рамку и эту часть вашего решения все равно никто не увидит и не сможет правильно оценить. Мы рекомендуем даже специально отступить от границ этой рамки примерно на полсантиметра, на всякий случай.

2. Все, что вы пишете в бланках ответа, обязательно пишите только черной гелевой ручкой. Даже рисунки, таблицы и графики, если вы рисуете их на бланках, нужно рисовать ей же. Сканер имеет достаточно высокий порог контрастности. Он не умеет передавать полутона. Только черное или белое. Не делайте никаких рисунков карандашом. Только черная гелевая ручка.

3. Пишите четко и разборчиво. Если вы этого не умеете или если знаете, что ваш почерк не очень хорошо читается – пишите печатными буквами. Если эксперт не сможет разобрать, что вы написали в ответе – пострадаете вы, а не эксперт. Пусть даже ваше решение было замечательным, правильным или гениальным. Если его нельзя будет прочитать – это все равно, что вы не решили задачу. В конце концов, от аккуратной, разборчиво написанной работы у эксперта сложится гораздо более приятное впечатление, чем от неряшливой, непонятной, с зачеркиваниями. За красиво оформленную и понятную работу эксперт в спорных случаях скорее поставит балл в большую сторону, чем за неаккуратную.

4. Помните, что ваша основная цель – верно решить предложенные задачи, а не продемонстрировать экспертам свои знания суперсовременных языков и технологий программирования. Даже если вы придумали сверхгениальное решение – подумайте трижды, как записать его понятным и корректным образом. Если вы нашли несколько решений одной и той же задачи – рекомендуем вам выбрать более простое и понятное. Например, если при решении задачи С4 вы используете сортировку, нет необходимости

использовать алгоритмы QuickSort или HeapSort. Их изучение в школе не предполагается, поэтому от вас никто этого не ждет. Просто напишите в этом месте сортировку "пузырьком". Помните, что за использование в решении сведений сверх школьной программы никаких дополнительных баллов не полагается, но в сложных алгоритмах вероятность ошибиться выше. Используйте в своих программах комментарии, чтобы пояснить тот или иной фрагмент кода. Иногда полезно до написания самой программы изложить на естественном языке алгоритм решения задачи. Тогда эксперту будет понятно, что именно он проверяет.

5. Постарайтесь "понравиться" эксперту. Это звучит странно и дико. Однако здесь действует то же правило, что и при обычном устном, очном экзамене. Постарайтесь оформить работу аккуратно, четко, логично. Если вы сумеете создать у эксперта положительное мнение о себе – это "сыграет вам на руку". При оценивании вашей работы эксперт руководствуется "Критериями оценивания" каждого задания. В них приведены различные варианты и типичные ошибки, которые вы можете совершить. Однако разработчик задания не может предусмотреть всех. Или их описание займет несколько страниц. Поэтому у экспертов иногда возникают затруднения, как следует оценить ваше решение, если оно не укладывается в предложенную схему. Если ваша работа понравится эксперту внешне – он подсознательно будет склонен поставить вам больший балл.

1. Задание С1

ЧТЕНИЕ ФРАГМЕНТА ПРОГРАММЫ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ И ИСПРАВЛЕНИЕ В НЕМ ЛОГИЧЕСКИХ ОШИБОК

Характеристика задания с развернутым ответом С1

В соответствии со спецификацией экзамена, задание С1 нацелено на проверку умения читать короткую (30–50 строк) простую программу на алгоритмическом языке (языке программирования) и умение искать и исправлять ошибки в небольшом фрагменте (10–20 строк) программы.

Типичное условие задания С1

Вам предлагается некоторая задача и вариант ее решения, в котором (сознательно) допущена ошибка. Требуется найти эту ошибку и указать, как нужно изменить программу, чтобы она верно решала поставленную задачу.

Задание относится к Повышенному уровню сложности. Это означает, что задачу должно решить примерно 40%–60% от всего количества сдающих экзамен.

Решение задачи предлагается на нескольких языках программирования.

Обратите внимание – в задании требуется найти смысловую, а не синтаксическую ошибку. Это значит, что не следует проверять предложенную программу на соответствие синтаксису языка программирования – ответ, который вы должны будете дать, состоит вовсе не в этом. Даже если вы обнаружите синтаксическую ошибку, скорее всего, это означает всего лишь, что вы недостаточно знаете синтаксис языка. Поэтому не тратьте свое драгоценное время на экзамене – просто ищите смысловую ошибку (ошибки).

Какие знания и умения требуются для решения задания С1

Так как это задание – единственное в части С, которое относится к повышенному уровню сложности (все остальные – к высокому), то и знания в этой части проверяются не самые высокие. То есть, очень маловероятно, чтобы в программе требовалось использовать массивы или даже циклы. Скорее всего, программа будет содержать некоторое количество условий. Смысловая ошибка в программе, состоящей из условий – это значит, что в программе не рассмотрен один или несколько случаев. Как искать ошибку в такой программе? Самый правильный путь – решить задачу (составить программу) самостоятельно, рассмотрев самостоятельно все возможные случаи, и соотнести случаи, рассмотренные в данной программе со случаями, которые получились у вас.

Замечание. Если в тексте задания написано, что входные данные (например, параметры уравнения) удовлетворяют некоторым ограничениям, то не нужно считать, что программа должна эти ограничения проверять. Входные данные считаются корректными. Эти ограничения обычно даются для того, чтобы упростить задачу, уменьшить количество случаев, которые необходимо рассмотреть.

Например, если в задании сказано, что для входных параметров a и b должно выполняться $a>0$, $b\neq 0$, то не нужно считать, что в программе пропущены именно эти условия, и указывать их как ту самую ошибку, которую необходимо найти. Нужно решать задачу, будучи уверенными, что эти условия обязательно выполняются.

Темы по информатике, которые следует повторить при подготовке к экзамену:

Понимание структуры программы.
Понятие о переменной и о типах данных.
Ввод-вывод переменных.
Условный оператор (ветвление).
Полная и неполная форма условного оператора.

Межпредметные связи:

Темы по математике, которые следует повторить при подготовке к экзамену:

Решение уравнений и неравенств с параметром (линейных, квадратичных, с модулем, дробных). Графики функций.

Рассмотрим типичные задания С1.

Уравнения и неравенства с параметром

В задачах этого типа требуется решить в общем виде уравнение или неравенство с параметрами. Так как выпускник средней школы должен уметь решать такие задачи в курсе математики, эти знания составители считают возможным требовать и на экзамене по информатике. В то же время, так как экзамен не по алгебре, предлагаемые задания не должны выходить на рамки простейших знаний математики. Нужно уметь решать уравнения и неравенства с параметрами (линейные, квадратичные, с модулем, дробные). Мы считаем, что вам известны следующие знания из математики:

1. Вы знаете, что такое "аргумент", "модуль", "числитель", "знаменатель", "дискриминант", "корень уравнения".

2. Решить уравнение (неравенство) – это значит указать все такие значения аргумента, при которых это уравнение (неравенство) имеет смысл. Или указать, что таких нет.

3. На ноль делить нельзя.

4. При делении (умножении) обеих частей неравенства на отрицательное число знак неравенства меняется на противоположный.

5. При умножении на ноль все выражение становится равным нулю.

Как вы понимаете, чтобы дать правильный ответ на задание С1, нужно уметь делать две вещи: не только уметь решать такие задания самостоятельно, но и уметь читать чужую программу и понимать, что делает каждая ее строчка.

Рассмотрим первое умение: как составить программу, решающую уравнения (неравенства) с параметром. В двух словах – нужно рассмотреть все возможные случаи и указать в каждом случае ответ. Лучше всего для этого, конечно, обратиться к соответствующим учебникам по математике. Если у вас нет таких под рукой (или, что хуже – вам лень это сделать), мы попробуем вкратце напомнить вам, как это делается.

Начнем с уравнений. Общий метод решения – перенести в одну (обычно правую) часть уравнения все, кроме аргумента, который оставить в левой части. Это и будет ответ.

Линейные уравнения

Пример – линейное уравнение ($ax+b=0$). Переносим b в правую часть (получаем $ax=-b$). Делим обе части на a (получаем $x=-b/a$). Считаем, что это и есть ответ.

Здесь, однако, выявляется первая неприятность – чтобы поделить обе части на a, нужно чтобы это самое a было не равно нулю. Вы скажете – "на ноль делить нельзя! Значит, такое a просто не может быть и наше решение верно". И будете неправы. На ноль делить действительно нельзя. Но чтобы решить это уравнение, вовсе и не нужно делить на ноль при $a=0$. Нужно просто понять, что наше уравнение ($ax+b=0$) при $a=0$ превращается в уравнение $b=0$. А найти нам нужно x. "Ну и что", скажете вы, "при чем тут b, если мы ищем x, а наше уравнение осталось и вовсе без этого самого x!? Значит, при $a=0$ нет решений". И опять окажетесь не правы, ибо мы должны "найти все такие значения аргумента, при которых уравнение имеет смысл". А наше уравнение превратилось в $b=0$. В каком случае оно имеет смысл? Как раз в том случае, если значение b *равно* 0. Получаем еще один случай. То есть, если $a=0$, то нужно проверить b. Если b тоже равно нулю, то уравнение имеет смысл. И не зависит от x. То есть, равенство верно для любых x. А если $b\neq0$, то уравнение не имеет смысл ни при каких x. То есть, в этом случае решений нет. Получаем решение:

```
Если a=0, то проверяем
    Если b=0, то
        Ответ "x – любое число",
    Иначе
        Ответ "Нет решений",
Иначе
    Ответ "x=", –b/a
```

Обратите внимание, "иначе" относится к последнему "если", у которого еще нет своего "иначе"!

Уравнения с модулем

При решении уравнений с модулем нужно помнить, что если вы довели ваше уравнение до вида $|x|=d$ (где d, возможно, какое-то выражение из параметров), то такое уравнение может иметь три возможных решения:

если $d>0$, то уравнение имеет два ответа: $x=d$ или $x=-d$;

если $d=0$, то уравнение имеет один ответ: $x=0$;

если $d<0$, то уравнение не имеет решений.

Напоминаем, что первый и второй случаи – это случаи разные (разное количество корней). Недопустимо (считается ошибкой) решение, при котором в случае $d=0$ программа выводит "x=0 или x=0".

Рассмотрим решение уравнения $a|x|=b$. Хотелось бы оставить в левой части только $|x|$. Тогда уравнение сведется к уже рассмотренному виду $|x|=d$. Для этого нужно обе части уравнения поделить на a. Однако это можно сделать только в случае, когда $a\neq0$. Тогда уравнение сводится к виду $|x|=b/a$. Для его решения нужно рассмотреть три случая в зависимости от знака правой части, как было показано выше.

Если же $a=0$, то уравнение превращается в выражение $0=b$. Оно имеет смысл в случае, когда $b=0$ (тогда подходит любое значение для x), в противном случае ни одно значение x не сделает равенство верным. Объединяем полученные рассуждения:

```
Если a=0, то проверяем
    Если b=0, то
        Ответ "x – любое число",
    Иначе
        Ответ "Нет решений",
Иначе
    Если b/a>0, то
        Ответ "x=", b/a, " или x=", –b/a
    Иначе проверяем
        Если b/a=0, то
            Ответ "x=0",
        Иначе
            Ответ "Нет решений"
```

Неравенства

Теперь вспомним, как решаются неравенства.

Особенностью решения неравенств является то, что ответом, как правило, является не определенное число, а целый числовой интервал или отрезок.

Темы, которые рекомендуется повторить в курсе математики: изменение знака неравенства при умножении/делении обеих частей на отрицательное число, метод интервалов.

Решим неравенство $(x–a)/(bx)\geq 0$.

Это, конечно, не самый простой пример для неравенств, но зато очень показательный.

Первое, что хочется сделать – сразу воспользоваться методом интервалов, найти точки, в которых числитель или знаменатель меняют знак, обозначить их на числовой прямой, нарисовать "волну", расставить знаки "+" и "–" и записать ответ.

Но этому мешает то, что значение параметра b может изменить знак неравенства, а значение параметра a может оказаться равным нулю, тогда "x" в числителе и знаменателе вообще может сократиться.

То есть, опять необходимо аккуратно рассмотреть все возможные случаи. Рекомендуем воспользоваться методом "разделяй и властвуй" – аккуратно сначала рассмотреть все особые, "мешающие" общему решению случаи, а потом применить метод интервалов.

Нужно рассмотреть все особые случаи одного параметра и при каждом из них рассмотреть случаи для другого. Порядок при этом обычно не важен. Хотя в данном случае удобнее всего сначала рассмотреть случай, когда $b=0$, чтобы к этому больше не возвращаться, так как при этом дробь сразу не имеет смысла.

Для параметра a особым случаем является равенство его нулю (тогда дробь можно будет "сократить" на x, если $x\neq 0$).

Для параметра b особыми случаями являются его положения относительно нуля (если $b=0$ дробь не имеет смысла, если $b>0$, ее можно просто сократить (умножить на b обе части неравенства), если $b<0$, ее тоже можно сократить, но при этом знак неравенства поменяется).

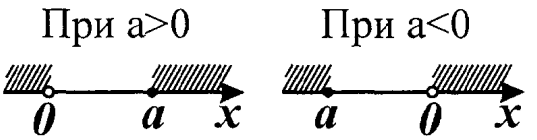
Итак, если $b\neq 0$, рассмотрим возможные значения a.

Если $a=0$, неравенство сводится к виду $x/(bx)\geq 0$ и тогда на x можно сократить (если $x\neq 0$), останется $1/b\geq 0$. Последнее верно при любых $x\neq 0$, если $b>0$, и не имеет смысла при $b<0$.

Если же $a\neq0$, то решение далее разделяется на два отдельных, непростых случая, когда $b>0$ и знак неравенства остается неизменным, и когда $b<0$ и знак неравенства меняется на противоположный.

Если $b>0$, неравенство сводится к виду $(x-a)/x\geq0$. Дробь меняет знак в точках 0 и a. Но чтобы обозначить эти точки на числовой прямой, нужно еще знать, как они между собой расположены. По методу интервалов нас интересуют точки вне отрезка, лежащего между a и 0.

Если $a>0$, точка a лежит справа от нуля (см. рисунок). Так как x находится в знаменателе, ноль не включается в ответ и решение при этом выглядит "x<0 или x≥a". Если же $a<0$, то точка a лежит слева от нуля и решение при этом "x≤a или x>0".



Если же $b<0$, то неравенство сводится к виду $(x-a)/x\leq0$. Теперь, по методу интервалов, нас интересуют точки внутри отрезка (точнее, полуотрезка), лежащего между a и 0. Но снова нужно проверить, что лежит правее – точки a или точка 0.

Если $a>0$, то решение "0<x≤a". Иначе решение "a≤x<0".

Формализуем все приведенные рассуждения:

```
Если b=0, то
    Ответ "Нет решений",
Иначе
    Если a=0, то
        Если b>0, то
            Ответ "Любой x≠0",
        Иначе
            Ответ "Нет решений",
    Иначе
        Если b>0, то
            Если a>0, то
                Ответ "x<0 или x≥a",
            Иначе
                Ответ "x≤a или x>0,
        Иначе
            Если a>0, то
                Ответ "0<x≤a",
            Иначе
                Ответ "a≤x<0"
```

Для тренировки решите, пожалуйста, следующие уравнения и неравенства с параметром:

1.1. $a/x=b$;

1.2. $a/|x|=b$,

1.3. $a|x|>b$;

1.4. $a|x|\leq b$;

1.5. $ax/(x-b)\leq0$.

Примеры решения заданий ЕГЭ
(уравнения и неравенства с параметрами)

Из демоверсии 2008 года:

Требовалось написать программу, которая решает уравнение «$a|x|=b$» относительно x для любых чисел a и b, введенных с клавиатуры. Все числа считаются действительными. Программист торопился и написал программу неправильно.

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var a,b,x: real; begin readln(a,b,x); if a=0 then if b=0 then write('любое число') else write('нет решений') else if b=0 then write('x=0') else write('x=',b/a, ' или x=',-b/a); end.	INPUT a, b, x IF a=0 THEN IF b=0 THEN PRINT "любое число" ELSE PRINT "нет решений" ENDIF ELSE IF b=0 THEN PRINT "x=0" ELSE PRINT "x=",b/a, " или x=",-b/a ENDIF ENDIF END	void main(void) { float a,b,x; scanf("%f%f%f", &a,&b,&x); if (a==0) if (b==0) printf("любое число"); else printf("нет решений"); else if (b==0) printf("x=0"); else printf("x=%f или x=%f", b/a,-b/a); }

Последовательно выполните три задания:

1) Приведите пример таких чисел a, b, x, при которых программа работает неправильно.

2) Укажите, какая часть программы является лишней.

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

Как мы уже писали, рекомендуется самостоятельно решить задачу и сравнить ее с приведенным решением (если вы сами не можете решить задачу, то и чужое решение вряд ли сможете понять). Решение этого уравнения ($a|x|=b$) мы рассмотрели несколькими страницами ранее. Соотнеся случаи, рассмотренные "программистом-неудачником, который вечно пишет неправильные задания С1 для ЕГЭ" с рассмотренными нами случаями, находим, что потерян случай, когда $b/a<0$.

Чтобы "доработать программу", как этого требует от нас условие задания, нужно перед тем, как выводить на экран ответ "x=–b/a или x=b/a", вставить проверку условия, что $b/a<0$. Если она выполнится – выводим "нет решений". Если не выполнится – выводим приведенный ответ.

Однако, чтобы получить желаемые три балла за эту задачу, мы должны четко ответить на три вопроса, которые нам задают. Мы рекомендуем отвечать на каждый вопрос отдельно, выделяя (например, подчеркиванием) свои ответы.

В первом вопросе нас просят указать те числа *a,b,x*, при которых программа работает неправильно. В этом вопросе не нужно писать, что программа неправильно работает при $b/a<0$. Напишите именно то, что у вас спрашивают – подберите такие *a* и *b*, при которых $b/a<0$ и укажите их здесь. Например, $a=1$, $b=-1$.

Однако, чтобы правильно ответить на первый вопрос, нужно не только решить задачу (рассмотреть все возможные случаи и найти пропущенный), но и найти ответ на второй вопрос – что в программе является лишним.

Если вы подошли к подготовке к экзамену правильно, то уже давно прорешали все демоверсии, выложенные на сайте ege.edu.ru, включая ДЕМО-2008, и сверили свои решения с приведенными там же ответами и критериями оценивания. И поэтому вы уже знаете правильный ответ.

Если же этого еще не случилось (а мы настоятельно рекомендуем сделать это), то мы заметим, что переменную "*x*" в этой задаче вовсе не нужно вводить с клавиатуры. Она не является входным параметром. Ее как раз необходимо найти. Поэтому лишним (и неправильным) фрагментом программы является ее ввод с клавиатуры. То есть, вводить нужно только *a* и *b*. Если еще немного подумать, то переменная "*x*" вообще в программе никак не используется и вовсе не нужна. Ее значение вычисляется сразу в момент вывода на экран.

Теперь мы готовы записать правильный ответ:

1. *Программа работает неверно при a=1, b=–1, x – не нужно вводить*

Вообще говоря, про "*x*" можно вообще не писать, ведь его ввод в программе является лишним. Но будет лучше, если вы покажете эксперту, какой вы старательный и что вы полностью понимаете задачу. Если первым пунктом просят указать такие *a,b,x*, то покажите, что про "*x*" вы забыли не по невнимательности, а сознательно.

2. *Не нужно вводить с клавиатуры переменную x (и описывать ее в разделе переменных тоже не нужно). Вместо readln(a,b,x); следует оставить только readln(a,b);*

Мы напоминаем, что отвечать на вопросы рекомендуется четко, именно то, что спрашивают. Не нужно считать, что если вы написали про "*x*" в ответе на первый вопрос, то можно не отвечать на второй. Не поленитесь! Это ведь ваш "+1 балл"! Также не рекомендуем ограничиваться ответом, что переменная "*x*" в программе совсем не нужна. Описание лишней переменной в разделе описаний – всего лишь недочет ("warning" , в терминах компилятора). А вот вводить с клавиатуры значение переменной, которое нужно найти – это уже ошибка. Если вы не напишете про то, что переменную "*x*" можно не описывать в программе – ничего страшного. А вот если вы не напишете, что ее не нужно вводить – можете потерять свой драгоценный балл.

3. *После строки с последним else нужно вставить строки*

```
if b/a<0 then
  write('нет решений')
else
```

В ответе на этот вопрос вы должны точно написать, какие строки программы и куда вы считаете нужным вставить. Если вы считаете, что не сможете точно объяснить, куда и что вы планируете вставить – просто приведите целиком пример своей правильно работающей программы.

Задания для самостоятельной работы

1.6. Требовалось написать программу, которая решает неравенство «$ax+b\leq 0$» относительно x для любых чисел a и b, введенных с клавиатуры. Все числа считаются действительными. Программист торопился и написал программу неправильно.

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var a,b,x: real; begin readln(a,b,x); if a=0 then if -b<=0 then write('любое число') else write('нет решений') else write('x>=',b/a); end.	INPUT a, b, x IF a=0 THEN IF -b<=0 THEN PRINT "любое число" ELSE PRINT "нет решений" ENDIF ELSE PRINT "x>=",b/a ENDIF END	void main(void) { float a,b,x; scanf("%f%f%f",&a,&b,&x); if (a==0) if(-b<=0) printf("любое число"); else printf("нет решений"); else printf("x>=%f",b/a); }

Последовательно выполните три задания:

1) Приведите пример таких чисел a, b, x, при которых программа работает неправильно.

2) Укажите, какая часть программы является лишней.

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

1.7. Требовалось написать программу, которая решает неравенство «$ax/(x-b)\leq 0$» относительно x для любого числа b и любого неотрицательного числа a ($a\geq 0$), введенных с клавиатуры. Все числа считаются действительными. Программист торопился и написал программу неправильно.

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var a,b,x: real; begin readln(a,b,x); if a=0 then write('x<>',b) else if b=0 then write('x<>0') else write('x>=0 или x<',b) end.	INPUT a, b, x IF a=0 THEN PRINT "x<>",b ELSE IF b=0 THEN PRINT "x<>0" ELSE PRINT "x>=0 или x<",b ENDIF ENDIF END	void main(void) { float a,b,x; scanf("%f%f%f",&a,&b,&x); if (a==0) printf("x<>%f",b); else if(b==0) printf("x<>0"); else printf("x>=0 или x<%f",b); }

Последовательно выполните три задания:

1) Приведите пример таких чисел *a*, *b*, *x*, при которых программа работает неправильно.

2) Укажите, какая часть программы является лишней.

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

Рассмотрим другой вид задания С1.

Определение принадлежности точки плоскости заштрихованной области на графике

В задачах этого вида дан рисунок координатной плоскости и некоей закрашенной/заштрихованной области, ограниченной набором прямых и кривых. Нужно определить, принадлежит ли точка с координатами (*x*,*y*) этой области или нет.

Так как уравнения всех границ указаны на графике, задачи этого вида требуют понимания того, какой знак нужно поставить в соответствующем неравенстве.

Выписав все неравенства, которые "ограничивают" закрашенную/заштрихованную область, и объединив их логической операцией «И» («AND»), получим условие, которое должна проверять программа. Если условие выполняется – точка принадлежит области, если не выполняется – не принадлежит.

Так, если искомую область ограничивает вертикальная прямая линия, то в общем виде уравнение этой прямой выглядит как "*x*=*a*", где *a* – то значение на оси *x*, через которое проходит данная прямая (напомним, что если эта прямая совпадает с осью *y*, то уравнение прямой будет "*x*=0").

Так как ось *x* направлена вправо (на что указывает "стрелка"), то все значения координаты *x* справа от нашей прямой будут больше числа *a*. А все значения координаты *x* слева от нашей прямой ("*x*=*a*") будут меньше *a*. Получаем, что если закрашенная/заштрихованная область лежит **справа** от прямой "*x*=*a*", то все ее точки удовлетворяют условию "*x*>=*a*", а если **слева** – то условию "*x*<=*a*".

Замечание относительно нестрогого неравенства (почему нужно писать именно "*x*>=*a*" вместо "*x*>*a*"): так как на графике все прямые и кривые нарисованы сплошной линией, то закрашенная/заштрихованная область включает свои границы. Поэтому мы и пишем "больше или равно" или "меньше или равно". Строгие неравенства (строго больше/меньше) нужно было бы использовать, если бы границы области были бы нарисованы пунктиром.

Все остальные прямые и кривые, ограничивающие область, уравнения которых сводятся к виду "*y=какое-то_выражение*" нужно рассматривать с позиции *y*-координаты (ведь это именно она стоит в левой части уравнения). Так как ось "*y*" направлена вверх, то все точки, которые лежат **выше** прямой/кривой, должны удовлетворять условию "*y>=это_выражение*", а все точки, лежащие **ниже** нее – условию "*y<=это_выражение*". Значит, если закрашенная/заштрихованная область лежит **выше** рассматриваемой прямой/кривой, то все ее точки удовлетворяют условию "*y>=это_выражение*", а если **ниже** – то условию " *y<=это_выражение*".

Напомним, что если нужная вам прямая горизонтальна, то ее уравнение имеет вид "*y*=*b*", где *b* – то значение на оси *y*, через которое проходит эта прямая. И, конечно, если прямая совпадает с осью *x*, то ее уравнение имеет вид "*y*=0".

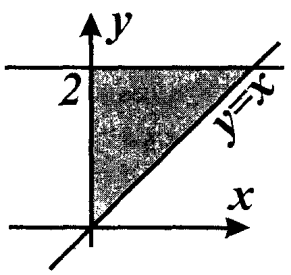
Замечание:

Вывод уравнения прямой, наклонной к оси *x* выходит за рамки данной книги. На ЕГЭ по информатике вам это умение не понадобится, т.к. это уравнение будет явно указано на графике. А если вам это интересно для общего развития – полистайте учебник алгебры за предыдущие годы.

Единственный и особый случай, который мы хотели бы рассмотреть и напомнить вам – когда закрашенную/заштрихованную область ограничивает окружность. Если ее центр совпадает с началом координат, то уравнение точек на окружности имеет вид "$x^2+y^2=R^2$", где R – радиус этой окружности.

Как вы, вероятно, понимаете, левая часть этого уравнения (x^2+y^2) задает квадрат расстояния от начала координат до точки плоскости. Для всех точек нашей окружности это расстояние равно ее радиусу R. Именно в этом смысл уравнения окружности. Значит, для всех точек **внутри** окружности, включая саму окружность, это расстояние будет меньше R. Это можно выразить неравенством "$x^2+y^2<=R^2$". А для всех точек **снаружи** окружности неравенство будет "$x^2+y^2>R^2$".

Рассмотрим пример:



В данной задаче закрашенную область ограничивают три прямые линии: горизонтальная (ее уравнение "$y=2$"), наклонная (ее уравнение "$y=x$") и вертикальная (она совпадает с осью y, ее уравнение "$x=0$").

Относительно горизонтальной прямой закрашенная область находится снизу, поэтому знак неравенства должен быть "меньше или равно" – "$y<=2$".

Относительно наклонной прямой закрашенная область находится сверху, поэтому знак неравенства должен быть "больше или равно" – "$y>=x$".

Относительно вертикальной прямой закрашенная область находится справа, поэтому знак неравенства должен быть "больше или равно" – "$x>=0$".

Итого получаем условие: (y<=2) И (y>=x) И (x>=0).

Соответственно,

на Паскале и на Бейсике:

(y<=2) AND (y>=x) AND (x>=0)

на Си:

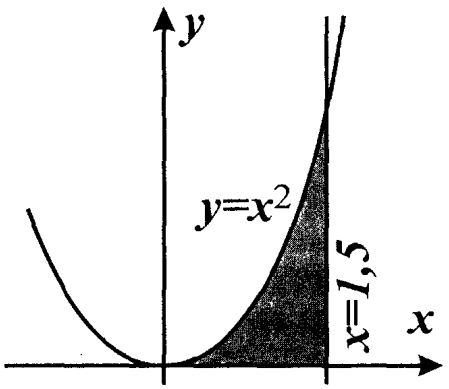
(y<=2) && (y>=x) && (x>=0)

На Бейсике и Си, в данном случае, скобки можно не писать. Но лучше приучиться писать их всегда, чтобы не задумываться каждый раз, нужны они или нет.

В этом примере задача была очень простой – закрашенная область была ограничена только прямыми линиями. Если же область ограничена кривыми (или хотя бы одной кривой), то в список условий обычно приходится добавить еще одно или два условия – ведь кривая может в стороне от закрашенной области изменять свое направление.

И тогда получится, что набора ограничений, который мы написали, просто перечислив линии, ограничивающие область, может оказаться недостаточно.

Рассмотрим пример:



Линий, ограничивающих закрашенную область, три: вертикальная прямая $x=1,5$, горизонтальная прямая $y=0$ и кривая $y=x^2$.

Руководствуясь вышеприведенными рассуждениями, строим систему условий:

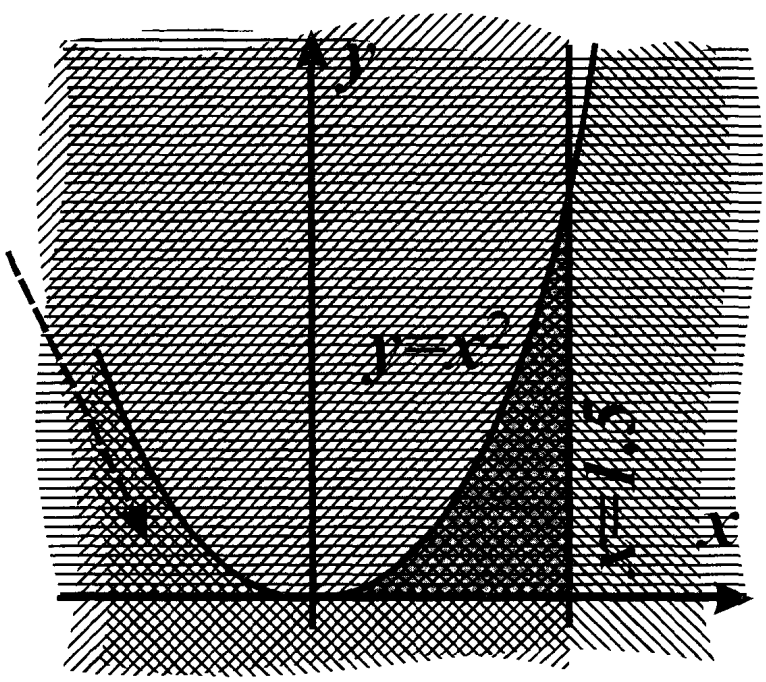
$x<=1.5$ (закрашенная область лежит левее вертикальной прямой),

$y>=0$ (закрашенная область лежит выше горизонтальной прямой) и

$y<=x^2$ (закрашенная область лежит ниже кривой).

Однако, этих условий недостаточно, чтобы ограничить закрашенную область.

Например, чтобы это понять, можно воспользоваться методом "штриховки": для каждого условия заштрихуем разными способами область плоскости, удовлетворяющую этому условию. Там, где все штриховки пересекутся (наложатся одна на другую), все условия будут выполняться.



На рисунке видно, что все три штриховки накладываются друг на друга не только на закрашенной области, но и в области, на которую указывает пунктирная стрелка. Этот эффект объясняется очень просто – правило, которое позволяет задать область

плоскости, просто перечислив уравнения ограничивающих линий с соответствующим знаком, действует только в том случае, если эти линии *не изменяют своего направления (т.е. функции, графиками которых они являются – возрастающие или убывающие).* В случае с параболой $y=x^2$ это не так – слева от начала координат функция $y=x^2$ убывает, а справа – возрастает, в результате чего наши условия неверно ограничивают закрашенную область.

Для исправления этой ошибки нужно добавить еще одно условие. Например, "x>=0". Таким образом, правильным ответом в данном случае будет:

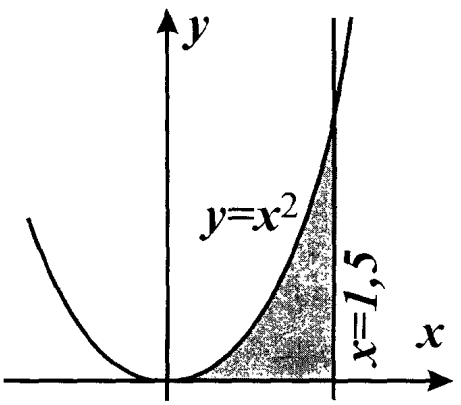
На Паскале и Бейсике:

(x>=0) AND (x<=1.5) AND (y>=0) AND (y<=x*x)

На Си:

(x>=0) && (x<=1.5) && (y>=0) && (y<=x*x)

Примеры решения заданий ЕГЭ
(принадлежность точки заштрихованной области)



Требовалось написать программу, которая вводит с клавиатуры координаты точки на плоскости (x, y – действительные числа) и определяет принадлежность точки заштрихованной области. Программист торопился и написал программу неправильно.

Последовательно выполните следующее:

1) Приведите пример таких чисел x, y при которых программа работает неправильно.

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

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var x,y: real; begin readln(x,y); if y<=x*x then if x<=1.5 then if y>=0 then write('принадлежит') else write('не принадлежит') end.	INPUT x, y IF y<=x*x THEN IF x<=1.5 THEN IF y>=0 THEN PRINT "принадлежит" ELSE PRINT "не принадлежит" ENDIF ENDIF ENDIF END	void main(void) { float x,y; scanf("%f%f",&x,&y); if (y<=x*x) if (x<=1.5) if (y>=0) printf("принадлежит"); else printf("не принадлежит"); }

Как мы только что обсудили, для задания закрашенной/заштрихованной области нужно использовать 4 условия:

$x \geq 0$, $x \leq 1{,}5$, $y \geq 0$ и $y \leq x^2$, которые должны выполниться одновременно.

Изучаем предложенную программу и замечаем, что одного условия – $x \geq 0$ – не хватает. Без него под систему условий $x \leq 1{,}5$, $y \geq 0$ и $y \leq x^2$ подпадают также точки под параболой слева от оси *OY* и сверху от оси *OX*. Чтобы ответить на первый вопрос задачи (и получить свой первый балл), вычислим какую-нибудь точку из этой области. Она должна быть сверху от оси *OX* (значит, $y \geq 0$), слева от оси *OY* (значит, $x \leq 0$) и ниже параболы (значит, $y \leq x^2$). Возьмем какую-нибудь точку на оси *OX* такую, что $x \leq 0$, например, $x = -1$. Для нее должно выполниться $y \geq 0$ и $y \leq x^2$. То есть нас устроит любое значение y от 0 до $(-1)^2$. Например, 0,5.

Отвечаем на первый вопрос: $x = -1$, $y = 0{,}5$

Теперь постараемся исправить программу. Основное (и очевидное), что нужно в программу добавить – проверку еще одного условия ($x \geq 0$). Давайте это сделаем, например, добавив еще один IF в программу (выделено **жирным**):

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
`var x,y: real;`	`INPUT x, y`	`void main(void)`
`begin`	`IF y<=x*x THEN`	`{ float x,y;`
`readln(x,y);`	`IF x<=1.5 THEN`	`scanf("%f%f",&x,&y);`
`if y<=x*x then`	`IF y>=0 THEN`	`if (y<=x*x)`
`if x<=1.5 then`	**`IF x>=0 THEN`**	`if (x<=1.5)`
`if y>=0 then`	`PRINT "принадлежит"`	`if (y>=0)`
`if x>=0 then`	`ELSE`	**`if (x>=0)`**
`write('принадлежит')`	`PRINT "не принадлежит"`	`printf("принадлежит");`
`else`	**`ENDIF`**	`else`
`write('не принадлежит')`	`ENDIF`	`printf("не принадлежит");`
`end.`	`ENDIF`	`}`
	`ENDIF`	
	`END`	

Однако за такое решение мы получим еще только один балл (из двух возможных).

Второй балл нам не дадут, потому что в программе "спрятана" не одна ошибка, а две – во-первых, программа не проверяет условие $x \leq 0$; во-вторых, система IF'ов, которая использована в программе, содержит только один ELSE. А он, как известно, относится только к последнему IF'у, у которого нет своего ELSE.

То есть, программа работает неправильно не только для вычисленной нами еще одной области, но и для всех случаев, когда условия первых двух IF'ов не выполняются (при $y > x^2$ и при $x > 1.5$). В этих случаях программа просто ничего не выдает в качестве ответа (а должна выдавать "не принадлежит").

Чтобы исправить эту ошибку нужно либо добавить еще столько ELSE, сколько не хватает (в нашем, исправленном, случае – еще три):

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var x,y: real; begin readln(x,y); if y<=x*x then if x<=1.5 then if y>=0 then **if x>=0 then** write('принадлежит') else write('не принадлежит') **else** **write('не принадлежит')** **else** **write('не принадлежит')** **else** **write('не принадлежит')** end.	INPUT x, y IF y<=x*x THEN IF x<=1.5 THEN IF y>=0 THEN **IF x>=0 THEN** PRINT "принадлежит" ELSE PRINT "не принадлежит" **ENDIF** **ELSE** **PRINT "не принадлежит"** ENDIF **ELSE** **PRINT "не принадлежит"** ENDIF **ELSE** **PRINT "не принадлежит"** ENDIF END	void main(void) { float x,y; scanf("%f%f",&x,&y); if (y<=x*x) if (x<=1.5) if (y>=0) **if (x>=0)** printf("принадлежит"); else printf("не принадлежит"); **else** **printf("не принадлежит");** **else** **printf("не принадлежит");** **else** **printf("не принадлежит");** }

Либо (что гораздо проще и короче) записать условие принадлежности точки области через сложное условие (через логическое И).

Рекомендуемый ответ должен выглядеть так:

Ответ:

1. $x = -1$, $y = 0{,}5$

2.

ПРОГРАММА НА ПАСКАЛЕ	ПРОГРАММА НА БЕЙСИКЕ	ПРОГРАММА НА СИ
var x,y: real; begin readln(x,y); if (y<=x*x) and (x<=1.5) and (y>=0) and (x>=0) then write('принадлежит') else write('не принадлежит') end.	INPUT x, y IF (y<=x*x) AND (x<=1.5) AND (y>=0) AND (x>=0) THEN PRINT "принадлежит" ELSE PRINT "не принадлежит" ENDIF END	void main(void) { float x,y; scanf("%f%f",&x,&y); if (y<=x*x && x<=1.5 && y>=0 && x>=0) printf("принадлежит"); else printf("не принадлежит"); }

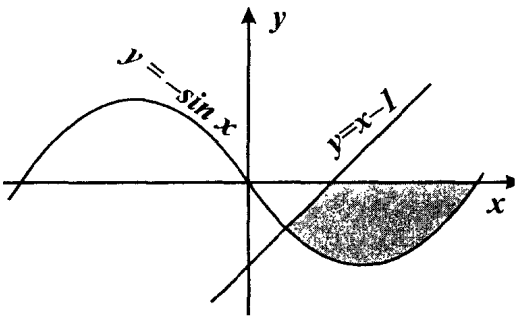
Естественно, вам нужно написать ответ только на одном языке программирования, наиболее вам известном и понятном.

Обращаем ваше внимание на то, что при записи на Паскале сложных условий, необходимо каждое условие заключить в скобки, т.к. приоритет операции AND выше, чем операции сравнения.

На языках Бейсик и Си каждое условие в скобки можно не заключать, т.к. приоритет операций <=, >= выше, чем у операций AND (Бейсик) и && (Си). Но если вы сомневаетесь, то лучше поставьте – ошибкой это не будет.

Задания для самостоятельной работы

1.8. Требовалось написать программу, которая вводит с клавиатуры координаты точки на плоскости (*x*,*y* – действительные числа) и определяет принадлежность точки закрашенной области, включая ее границы. Программист торопился и написал программу неправильно.



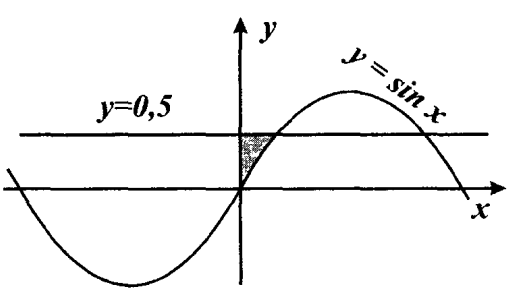
Бейсик	Паскаль	Си
INPUT x, y IF y<=x-1 THEN IF y<=0 THEN IF y>= -SIN(x) THEN PRINT "принадлежит" ELSE PRINT "не принадлежит" ENDIF ENDIF ENDIF END	var x,y: real; begin readln(x,y); if y<=x-1 then if y<=0 then if y>= -sin(x) then write('принадлежит') else write('не принадлежит') end.	void main(void) { float x,y; scanf("%f%f",&x,&y); if (y<=x-1) if (y<=0) if (y>= -sin(x)) printf("принадлежит"); else printf("не принадлежит"); }

Последовательно выполните следующее:

1) Приведите пример таких чисел *x*, *y*, при которых программа неверно решает поставленную задачу.

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

1.9. Требовалось написать программу, которая вводит с клавиатуры координаты точки на плоскости (*x*, *y* – действительные числа) и определяет принадлежность точки закрашенной области, включая ее границы. Программист торопился и написал программу неправильно.



Бейсик	Паскаль	Си
INPUT x, y IF x>=0 THEN IF y<=0.5 THEN IF y>= SIN(x) THEN PRINT "принадлежит" ELSE PRINT "не принадлежит" ENDIF ENDIF ENDIF END	var x,y: real; begin readln(x,y); if x>=0 then if y<=0.5 then if y>= sin(x) then write('принадлежит') else write('не принадлежит') end.	void main(void) { float x,y; scanf("%f%f",&x,&y); if (x>=0) if (y<=0.5) if (y>= sin(x)) printf("принадлежит"); else printf("не принадлежит"); }

Последовательно выполните следующее:

1) Приведите пример таких чисел *x*, *y*, при которых программа неверно решает поставленную задачу.

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

2. Задание С2

СОЗДАНИЕ КОРОТКОЙ ПРОСТОЙ ПРОГРАММЫ ИЛИ АЛГОРИТМА, ЗАПИСАННОГО НА ЕСТЕСТВЕННОМ ЯЗЫКЕ

Характеристика задания с развернутым ответом С2

В соответствии со спецификацией экзамена, задание С2 нацелено на проверку умения написать короткую (10–15 строк) простую программу (например, обработки массива) на языке программирования или записать алгоритм на естественном языке.

Примеры возможных задач по кодификатору (список не является исчерпывающим):

- Нахождение минимума и максимума двух, трех, четырех данных чисел без использования массивов и циклов.
- Нахождение всех корней заданного квадратного уравнения.
- Запись натурального числа в позиционной системе с основанием меньшим или равным 10. Обработка и преобразование такой записи числа.
- Использование цикла для решения простых переборных задач (поиск наименьшего простого делителя данного натурального числа, проверка числа на простоту, и т.д.)
- Нахождение сумм, произведений элементов данной конечной числовой последовательности (или массива).
- Заполнение элементов одномерного и двумерного массива по заданным правилам.
- Операции с элементами массива. Линейный поиск элемента. Вставка и удаление элементов в массиве. Перестановка элементов данного массива в обратном порядке. Суммирование элементов массива. Проверка соответствия элементов массива некоторому условию.
- Нахождение второго по величине (второго максимального или второго минимального) значения в данном массиве за однократный просмотр массива.
- Нахождение минимального (максимального) значения в данном массиве и количества элементов, равных ему за однократный просмотр массива.
- Операции с элементами массива, отобранных по некоторому условию (например, нахождение минимального четного элемента в массиве, нахождение количества и суммы всех четных элементов в массиве).
- Сортировка массива.
- Слияние двух упорядоченных массивов в один без использования сортировки.
- Обработка отдельных символов данной строки. Подсчет частоты появления символа в строке.
- Работа с подстроками данной строки с разбиением на слова по пробельным символам. Поиск подстроки внутри данной строки, замена найденной подстроки на другую строку.

Заметим, что в 2010 году требования к заданиям С2 были расширены. Ранее в задаче С2 предполагалось написать программу обработки массива. С 2010 года в задаче С2 возможны не только программы обработки массива, но и некоторые другие программы, в частности – программы линейной обработки входной последовательности.

Мы разобьем обсуждение данных заданий на четыре этапа. На первом этапе мы обсудим программы, не требующие циклов (только ветвления). На втором этапе – программы, требующие использования цикла, но не требующие работы с массивами. На третьем этапе разберем обработку массивов. И на четвертом этапе обсудим обработку символьных строк.

Отметим, что хотя это задание разрешается написать на естественном языке (читай "русском"), его выполнение требует знания понятия массива и понимания принципов работы с ними. В отличие от задания С1, это и следующие задания относятся к Высокому уровню сложности.

Всего за это задание можно получить только 2 балла. Поэтому если вы по каким-либо причинам не успеваете сделать все задания части С или не слишком сильны в программировании (мы предполагаем, что все задания частей А и В вы, все же, сделаете) – рекомендуем сначала уделить время задаче С3, а потом вернуться к задаче С2.

Как только мы начинаем писать программы хоть немного сложные, сразу нужно беспокоиться о том, правильно ли работает написанная нами программа. Единственный и самый правильный способ проверить это самостоятельно – выполнить трассировку программы.

Трассировка

Трассировка – пошаговое выполнение алгоритма с отслеживанием значений всех переменных. Если вы записываете алгоритм или программу на бумаге, то трассировка – единственный способ проверить их правильность.

Трассировка бывает подробной и упрощенной ("упрощенную" можно делать на бумаге и в уме).

Профессиональные (и самоуверенные) программисты совершают трассировку в уме. Более ответственные – совершают упрощенную трассировку на бумаге. Мы рекомендуем вам сначала освоить подробную трассировку, а потом пользоваться упрощенной.

Подробная трассировка – таблица, в которой выписываются по порядку выполнения все строки проверяемого алгоритма/программы, результаты проверки условий и изменившиеся значения всех переменных.

Рассмотрим пример подробной трассировки программы вычисления наибольшего общего делителя двух натуральных чисел.

Программа:

На Паскале:	На Си:	На Бейсике:
`readln(a,b);` `while a<>b do` `  if a>b then` `  a:=a-b` `  else` `  b:=b-a;` `writeln(a);`	`scanf("%d %d",&a,&b);` `while(a!=b)` `  if(a>b)` `  a-=b;` `  else` `  b-=a;` `printf("%d",a);`	`INPUT a,b` `WHILE a <> b` `IF a > b THEN` `a = a - b` `ELSE` `b = b - a` `ENDIF` `WEND` `PRINT a`

Выполним трассировку алгоритма для a=12, b=8.

Для этого построим таблицу.

В первом столбце пишем строку (оператор) программы, которая в данный момент выполняется. В следующем столбце записываем проверяемое условие (если в строке проверяется условие) и результат проверки (Да или Нет). Далее нужно выделить по одному столбцу для каждой переменной, которая используется в программе и записывать в ячейку новое значение переменной (если она меняется). В последнем столбце записываются результаты вывода на экран.

В нашей таблице трассировки вместо одного столбца со строками/операторами программы – три (для разных языков программирования). Естественно, вам нужно будет писать только один.

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		A	B	
`readln(a,b);`	`scanf("%d d",&a,&b);`	`INPUT A,B`		12	8	
`while a<>b do`	`while(a!=b)`	`WHILE A<>B`	12≠8, Да			
`if a>b then`	`if(a>b)`	`IF A>B THEN`	12>8, Да			
`a:=a-b`	`a-=b;`	`A = A - B`		4		
`while a<>b do`	`while(a!=b)`	`WHILE A<>B`	4≠8, Да			
`if a>b then`	`if(a>b)`	`IF A>B THEN`	4>8, Нет			
`b:=b-a`	`b-=a;`	`B = B - A`			4	
`while a<>b do`	`while(a!=b)`	`WHILE A<>B`	4≠4, Нет			
`writeln(a);`	`printf("%d",a);`	`PRINT A`				4

Мы рассмотрели пример трассировки правильно написанной программы. Мы ее выполнили и убедились, что все именно так, как мы и предполагали. Считаем, что программа написана правильно.

Замечание

Конечно же, одной проверки в общем случае недостаточно. Нужно проверить написанную программу на различные частные/граничные случаи. Например, для данной программы нужно также убедиться, что она правильно работает для изначально равных чисел a и b. И еще, неплохо бы, проверить программу для случая, когда одно из чисел равно единице.

Мы не останавливаемся подробно на обсуждении того, на каких входных данных нужно проверять программу, потому что это – большое искусство и его обсуждение выходит за рамки нашей книги.

Очень важно при трассировке не принимать желаемое за действительное. Трассировку нужно делать, прежде всего, не для того, чтобы убедиться в работоспособности программы. Наоборот, – для того, чтобы найти ошибку.

Нужно выбрать входные данные и четко представить себе, как должна (по вашему мнению) работать программа при этих входных данных.

После этого нужно аккуратно, пунктуально, беспристрастно, выполнить свою программу по шагам. Самая распространенная ошибка при этом – считать, что про-

грамма написана правильно. Наоборот, нужно не доверять самому себе и "тупо" выполнить трассировку. Если при этом окажется, что программа делает не совсем то, что по вашим планам она должна в этот момент делать – очень хорошо. Значит, Вы нашли ошибку. Проанализируйте, почему программа не делает то, что Вы планируете, исправьте ошибку, и снова выполните трассировку.

Рассмотрим пример использования трассировки для нахождения ошибок в программе.

Задача – программа должна вводить с клавиатуры целые числа до тех пор, пока введенное число не будет отрицательным, и вывести на экран сумму тех чисел, которые кратны трем. Предположим, первая версия программы выглядит так (кстати, это очень распространенный случай написания такой программы для начинающих):

На Паскале:	На Си:	На Бейсике:
`while a<0 do` `begin` `  readln(a);` `  if a mod 3=0 then` `  s:=s+a;` `end;` `writeln(a);`	`while(a<0)` `{` `  scanf("%d",&a);` `  if(a%3==0)` `  s+=a;` `}` `printf("%d",a);`	`WHILE a < 0` `INPUT a` `IF a MOD 3 = 0 THEN` `s = s + a` `ENDIF` `WEND` `PRINT a`

Выполним трассировку программы для последовательности вводимых чисел: 6, 5, –6.

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
`while a<0 do`	`while(a<0)`	`WHILE a<0`	???<0			

Попытка выполнить первую же строку программы наталкивается на непонимание – "Чему равно значение переменной *a*, если в столбце "переменная *a*" ничего не написано?" Распространенная ошибка при этом – считать, что значение переменной *a* равно 6 (ведь именно это значение мы решили первым подать на вход программе).

Правильное рассуждение – остановить трассировку и обрадоваться, что мы нашли первую ошибку. Оказывается, мы неверно предположили, что к моменту начала цикла while значение переменной *a* будет известно. Исправим этот недостаток – добавим ввод переменной *a* перед циклом:

На Паскале:	На Си:	На Бейсике:
`readln(a);` `while a<0 do` `begin` `  readln(a);` `  if a mod 3=0 then` `  s:=s+a;` `end;` `writeln(a);`	`scanf("%d",&a);` `while(a<0)` `{` `  scanf("%d",&a);` `  if(a%3==0)` `  s+=a;` `}` `printf("%d",a);`	`INPUT a` `WHILE a < 0` `INPUT a` `IF a MOD 3 = 0 THEN` `s = s + a` `ENDIF` `WEND` `PRINT a`

Снова выполним трассировку программы для тех же входных значений:

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
`readln(a);`	`scanf("%d",&a);`	`INPUT a`		6		
`while a<0 do`	`while(a<0)`	`WHILE a<0`	6<0, Нет			

И снова при попытке проверить условие цикла while мы наталкиваемся на противоречие с представлениями о том, как должна работать наша программа.

Мы предполагали, что наш цикл должен остановиться при вводе с клавиатуры отрицательного числа. Поэтому написали в while условие a<0. При трассировке оказывается, что для введенного числа 6 цикл сразу закончится и не выполнится ни разу. Мы вспоминаем, что в цикле while нужно писать условие выполнения цикла, а условие, которое мы написали (a<0) – условие его завершения. Значит, нужно поменять условие на противоположное[1]. Новая версия программы:

На Паскале:	На Си:	На Бейсике:
`readln(a);` `while a>=0 do` `begin` `  readln(a);` `  if a mod 3=0 then` `  s:=s+a;` `end;` `writeln(a);`	`scanf("%d",&a);` `while(a>=0)` `{` `  scanf("%d",&a);` `  if(a%3==0)` `  s+=a;` `}` `printf("%d",a);`	`INPUT a` `WHILE a >= 0` `INPUT a` `IF a MOD 3 = 0 THEN` `s = s + a` `ENDIF` `WEND` `PRINT a`

Снова выполняем трассировку:

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
`readln(a);`	`scanf("%d",&a);`	`INPUT a`		6		
`while a>=0 do`	`while(a>=0)`	`WHILE a >= 0`	6≥0, Да			
`readln(a);`	`scanf("%d",&a);`	`INPUT a`		5		
`if a mod 3=0 then`	`if(a%3==0)`	`IF a MOD 3 = 0 THEN`	2=0, Нет			

Примерно в этот момент (а лучше – строчкой раньше) мы обнаруживаем, что первое число (6), которое было введено с клавиатуры, не проверяется на кратность трем и поэтому не будет добавлено в сумму. Анализируем, почему это произошло и замечаем, что число 6 "пропадет" из-за того, что первым действием в цикле будет ввод нового значения переменной *a*. Делаем вывод, что снова вводить переменную *a* в начале цикла не нужно – ее значение уже введено перед циклом, и нужно его теперь проверять на кратность трем.

[1] Распространенная ошибка — заменить знак неравенства на противоположный. Правильно изменить также строгость неравенства. В данном случае знак "меньше" (<) должен быть заменен на знак "больше или равно" (>=).

Удаляем ввод переменной *a* в начале цикла:

На Паскале:	На Си:	На Бейсике:
`readln(a);` `while a>=0 do` `begin` `  if a mod 3=0 then` `  s:=s+a;` `end;` `writeln(a);`	`scanf("%d",&a);` `while(a>=0)` `{` `  if(a%3==0)` `  s+=a;` `}` `printf("%d",&a);`	`INPUT a` `WHILE a >= 0` `IF a MOD 3 = 0 THEN` `s = s + a` `ENDIF` `WEND` `PRINT a`

Выполняем трассировку получившейся программы:

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
`readln(a);`	`scanf("%d",&a);`	`INPUT a`		6		
`while a>=0 do`	`while(a>=0)`	`WHILE a >= 0`	6≥0, Да			
`if a mod 3=0 then`	`if(a%3==0)`	`IF a MOD 3 = 0 THEN`	0=0, Да			
`s:=s+a;`	`s+=a;`	`s = s + a`			??	

Если мы достаточно аккуратны, то в этот момент мы должны обнаружить, что при добавлении значения переменной *a* к значению переменной *s* нужно знать значение переменной *s*. А в столбце *s* таблицы трассировки у нас ничего не написано. Значит, переменная *s* к этому моменту не определена. Мы нашли еще одну ошибку – забыли обнулить переменную для суммы. Исправляем ошибку:

На Паскале:	На Си:	На Бейсике:
`s:=0;` `readln(a);` `while a>=0 do` `begin` `  if a mod 3=0 then` `  s:=s+a;` `end;` `writeln(a);`	`s=0;` `scanf("%d",&a);` `while(a>=0)` `{` `  if(a%3==0)` `  s+=a;` `}` `printf("%d",a);`	`s = 0` `INPUT a` `WHILE a >= 0` `IF a MOD 3 = 0 THEN` `s = s + a` `ENDIF` `WEND` `PRINT a`

Выполняем трассировку:

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
`s:=0;`	`s=0;`	`s = 0`			0	
`readln(a);`	`scanf("%d",&a);`	`INPUT a`		6		
`while a>=0 do`	`while(a>=0)`	`WHILE a >= 0`	6≥0, Да			
`if a mod 3=0 then`	`if(a%3==0)`	`IF a MOD 3 = 0 THEN`	0=0, Да			
`s:=s+a;`	`s+=a;`	`s = s + a`			6	
`while a>=0 do`	`while(a>=0)`	`WHILE a >= 0`	6≥0, Да			
`if a mod 3=0 then`	`if(a%3==0)`	`IF a MOD 3 = 0 THEN`	0=0, Да			
`s:=s+a;`	`s+=a;`	`s = s + a`			12	

В этот момент, опять-таки, важно не принимать желаемое за действительное и не считать, что на новом шаге цикла будет анализироваться новое значение переменной *a*. Важно заметить, что значение переменной *a* не изменилась, что оно продолжает оставаться равным 6 и что это значение мы уже проверяли на кратность трем и уже добавляли к сумме.

Мы понимаем, что удаление из начала цикла ввода переменной *a* было неверным. В начале цикла вводить переменную *a* было не нужно. Но это не значит, что в цикле не должно быть вообще ввода. Мы понимаем, что если программа должна вводить много чисел, то ввод должен быть в цикле. Однако, имеющаяся у нас программа прекрасно справляется с первым введенным числом – корректно проверяет условие выполнения цикла, проверяет кратность и увеличивает сумму. Значит, ввод переменной *a* нужно вписать в такое место, чтобы это значение проверялось на втором шаге цикла – вписываем его последней строчкой цикла:

На Паскале:	На Си:	На Бейсике:
s:=0; readln(a); while a>=0 do begin if a mod 3=0 then s:=s+a; readln(a); end; writeln(a);	s=0; scanf("%d",&a); while(a>=0) { if(a%3==0) s+=a; scanf("%d",&a); } printf("%d",a);	s = 0 INPUT a WHILE a >= 0 IF a MOD 3 = 0 THEN s = s + a ENDIF INPUT a WEND PRINT a

Выполняем трассировку:

Оператор программы			Условие	Переменные		На экране
На Паскале:	На Си:	На Бейсике:		a	s	
s:=0;	s=0;	s = 0			0	
readln(a);	scanf("%d",&a);	INPUT a		6		
while a>=0 do	while(a>=0)	WHILE a >= 0	6≥0, Да			
if a mod 3=0 then	if(a%3==0)	IF a MOD 3 = 0 THEN	0=0, Да			
s:=s+a;	s+=a;	s = s + a			6	
readln(a);	scanf("%d",&a);	INPUT a		5		
while a>=0 do	while(a>=0)	WHILE a >= 0	5≥0, Да			
if a mod 3=0 then	if(a%3==0)	IF a MOD 3 = 0 THEN	2=0, Да			
s:=s+a;	s+=a;	s = s + a			6	
readln(a);	scanf("%d",&a);	INPUT a		–6		
while a>=0 do	while(a>=0)	WHILE a >= 0	–6≥0,Нет			
writeln(a);	printf("%d",&a);	PRINT a				–6

В этот момент мы обнаруживаем, что программа все сделала правильно, но неправильный самый последний оператор – вместо значения переменной *s* (суммы) программа выводит значение переменной *a*. Исправляем это "недоразумение". Так как мы понимаем, что это исправление повлияет только на последнее действие программы, трассировку в этот раз уже не делаем – достаточно представить себе, что вместо:

writeln(a);	printf("%d",a);	PRINT a				-6

выполнится:

writeln(s);	printf("%d",s);	PRINT s				6

Когда Вы освоите подробную трассировку, можете использовать упрощенную.

При выполнении упрощенной трассировки выполняемая строка программы не записывается, а "отслеживается". В таблице трассировки пишут только столбцы для каждой переменной и столбец "на экране". Выполняемая строка программы и проверяемое условие вычисляются в уме.

Чтобы не запутаться, зачастую по строкам программы водят пальцем левой руки, а рядом, правой рукой, в таблице записывают значения изменившихся переменных ☺.

Профессионалы значения переменных тоже держат в уме.

Простые алгоритмы ветвления

При рассмотрении задач считаем, что нужные переменные описаны и инициализированы (введены с клавиатуры). Если по смыслу задачи тип данных неясен, считаем, что переменные описаны как вещественные. Если по смыслу задачи ясно, что числа целые, считаем, что переменные описаны как целые.

Нахождение максимума двух чисел

Самая простая задача, решаемая с помощью алгоритмической конструкции ветвления. Сравниваем два числа. Если первое оказывается больше второго, выводим первое. Иначе – выводим второе. Если числа окажутся равны, то можно вывести любое. В нашем случае выполнится ветвь алгоритма "иначе" и выведется второе число:

На Паскале:	На Си:	На Бейсике:
if a>b then write(a) else write(b);	if(a>b) printf("%f",a); else printf("%f",b);	IF A>B THEN PRINT A ELSE PRINT B ENDIF

Задания для самостоятельного решения

2.1. Выведите на экран меньшее из двух чисел.

Нахождение максимума трех чисел

Задача может быть решена несколькими способами. Самый простой – с использованием дополнительной переменной. Сравниваем первые два числа между собой и запоминаем большее в дополнительной переменной. Затем сравниваем третье число со значением этой переменной и, если третье число оказывается больше, выводим его. Если нет – выводим значение дополнительной переменной:

На Паскале:	На Си:	На Бейсике:
`if a>b then` `  tmp:=a` `else` `  tmp:=b;` `if c>tmp then` `  write(c)` `else` `  write(tmp);`	`if(a>b)` `  tmp=a;` `else` `  tmp=b;` `if(c>tmp)` `  printf("%f",c);` `else` `  printf("%f",tmp);`	`IF A>B THEN` `TMP=A` `ELSE` `TMP=B` `ENDIF` `IF C>TMP THEN` `PRINT C` `ELSE` `PRINT TMP` `ENDIF`

Самый понятный для начинающих (и самый неэффективный способ) – перебрать все возможные случаи большего числа:

На Паскале:	На Си:	На Бейсике:
`if (a>b) and (a>c) then` `  write(a)` `else` `  if (b>a) and (b>c) then` `  write(b)` `  else` `  if (c>a) and (c>b) then` `  write(c);`	`if(a>b && a>c)` `  printf("%f",a);` `else` `  if(b>a && b>c)` `  printf("%f",b);` `  else` `  if(c>a && c>b)` `  printf("%f",c);`	`IF A>B AND A>C THEN` `PRINT A` `ELSE` `IF B>A AND B>C THEN` `PRINT B` `ELSE` `IF C>A AND C>B THEN` `PRINT C` `ENDIF` `ENDIF` `ENDIF`

Заметим, что если числа равны, то программа не напечатает ничего. А если эту ошибку исправить (заменить знаки ">" на ">=", но убрать ELSE (распространенная ошибка начинающих)), то при одинаковых числах среди трех исходных программа будет выводить эти числа по несколько раз.

Этот метод можно несколько упростить, если заметить, что в случае невыполнения первого условия (A>B и A>C) переменная A уже не может оказаться самой большой и остается только выбрать между B и C:

На Паскале:	На Си:	На Бейсике:
`if (a>b) and (a>c) then` `  write(a)` `else` `  if b>c then` `  write(b)` `  else` `  write(c);`	`if(a>b && a>c)` `  printf("%f",a);` `else` `  if(b>c)` `  printf("%f",b);` `  else` `  printf("%f",c);`	`IF A>B AND A>C THEN` `PRINT A` `ELSE` `IF B>C THEN` `PRINT B` `ELSE` `PRINT C` `ENDIF` `ENDIF`

Однако, есть еще более эффективный алгоритм:

На Паскале:	На Си:	На Бейсике:
`if a>b then` `  if a>c then` `   write(a)` `  else` `   write(c)` `else` `  if b>c then` `    write(b)` `  else` `    write(c);`	`if(a>b)` `  if(a>c)` `    printf("%f",a);` `  else` `    printf("%f",c);` `else` `  if(b>c)` `    printf("%f",b);` `  else` `    printf("%f",c);`	`IF A>B THEN` `IF A>C THEN` `PRINT A` `ELSE` `PRINT C` `ENDIF` `ELSE` `IF B>C THEN` `PRINT B` `ELSE` `PRINT C` `ENDIF` `ENDIF`

Задания для самостоятельного решения

2.2. Выведите на экран меньшее из трех чисел.

2.3. Выведите на экран среднее из трех чисел (число, которое больше одного, но меньше другого).

2.4. Выведите на экран три числа в порядке возрастания.

2.5. Выведите на экран номер большего из трех чисел (1, 2 или 3).

Нахождение максимума четырех чисел

Задача может быть решена несколькими способами. Мы остановимся на самом простом – с использованием дополнительной переменной. Сравниваем первые два числа между собой и запоминаем большее в дополнительной переменной. Затем сравниваем третье число со значением этой переменной и, если третье число оказывается больше, кладем его значение в эту дополнительную переменную. Затем сравниваем четвертое число со значением дополнительной переменной и, если четвертое число оказывается больше, выводим его. Если нет – выводим значение дополнительной переменной:

На Паскале:	На Си:	На Бейсике:
`if a>b then` `  tmp:=a` `else` `  tmp:=b;` `if c>tmp then` `  tmp:=c;` `if d>tmp then` `  write(d)` `else` `  write(tmp);`	`if(a>b)` `  tmp=a;` `else` `  tmp=b;` `if(c>tmp)` `  tmp=c;` `if(d>tmp)` `  printf("%f",d);` `else` `  printf("%f",tmp);`	`IF A>B THEN` `TMP=A` `ELSE` `TMP=B` `ENDIF` `IF C>TMP THEN` `TMP=C` `ENDIF` `IF D>TMP THEN` `PRINT D` `ELSE` `PRINT TMP` `ENDIF`

Задания для самостоятельного решения

2.6. Выведите на экран меньшее из четырех чисел.

2.7. Выведите на экран номер большего из четырех чисел (1, 2, 3 или 4).

Нахождение всех корней заданного квадратного уравнения

Классическая задача. Решается классическим способом, через вычисление дискриминанта и анализ его значения. Будем считать, что уравнение имеет вид: $ax^2+bx+c=0$, где a, b, c – параметры. Однако, перед вычислением дискриминанта нужно убедиться, что решаемое уравнение – действительно квадратное ($a\neq0$). Если $a=0$, то уравнение превращается в линейное и нужно решать его:

На Паскале:

```
if a=0 then
 if b=0 then
  if c=0 then
   write('x - любое')
  else
   write('нет решений')
 else
  write('x=',-b/c)
else
begin
 D:=b*b-4*a*c;
 if D<0 then
  write('нет решений')
 else
  if D=0 then
   write('x=',-b/(2*a))
  else
   write('x=',
    (-b+sqrt(D))/(2*a),
    ' или x=',
    (-b-sqrt(D))/(2*a));
end;
```

На Си:

```
if(a==0)
 if(b==0)
  if(c==0)
   printf("x - любое");
  else
   printf("нет решений");
 else
  printf("x=%f",-b/c);
else
{
 D=b*b-4*a*c;
 if(D<0)
  printf("нет решений");
 else
  if(D==0)
   printf("x=%f",
          -b/(2*a));
  else
   printf("x=%f "
          "или x=%f",
    (-b+sqrt(D))/(2*a),
    (-b-sqrt(D))/(2*a));
}
```

На Бейсике:

```
IF A=0 THEN
IF B=0 THEN
IF C=0 THEN
PRINT "x - любое"
ELSE
PRINT "нет решений"
ENDIF
ELSE
PRINT "x=",-B/C
ENDIF
ELSE
D=B*B-4*A*C
IF D<0 THEN
PRINT "нет решений"
ELSE
IF D=0 THEN
PRINT "x=",-B/(2*A)
ELSE
PRINT "x=",
(-b+SQRT(D))/(2*a),
" или x=",
(-b-SQRT(D))/(2*a),
ENDIF
ENDIF
ENDIF
```

Задания для самостоятельного решения

2.8. Решите уравнение $ax^4+bx^2+c=0$.

2.9. Решите неравенство $ax^2+bx+c\geq0$.

Запись натурального числа в позиционной системе с основанием меньшим или равным 10

Для решения задачи достаточно воспользоваться классическим алгоритмом перевода числа из десятичной системы счисления в любую другую: делить нацело с остатком исходное число на основание нужной системы счисления пока исходное число не станет равно нулю и выписать полученные остатки в обратном порядке.

Единственная сложность – в обратном порядке выписывания остатков.

Если решать задачу без использования рекуррентных (рекурсивных) процедур, нужно остатки где-то хранить, чтобы после того, как исходное число стало равно нулю, выписать их в обратном порядке.

Самый простой способ (хотя и не самый эффективный) – складывать получающиеся остатки сразу в строковую переменную, причем каждый последующий остаток

дописывать к левой части строки. В результате в строке будет содержаться нужная нам запись числа. Будем считать, что исходное число – x, основание нужной системы счисления – a. Остатки будем хранить в строковой переменной s. Еще учтем, что получающиеся остатки – целые числа, а в строке должны храниться символы. Поэтому нужно преобразовывать остатки в символьный вид:

На Паскале:	На Си:	На Бейсике:
`if x=0 then` `  s:='0'` `else` `begin` `  s:='';` `  while x>0 do` `  begin` `  s:=chr(x mod a+` `  ord('0'))+s;` `  x:=x div a` `  end` `end;` `write(s);`	`int s[64];` `if(x==0)` `  printf("0");` `else` `{` `  k=0;` `  while(x>0)` `  {` `  s[k]=x%a;` `  k++;` `  x/=a;` `  }` `  for(i=k-1 ; i>=0 ; i--)` `  printf("%d",s[i]);` `}`	`IF X=0 THEN` `S$="0"` `ELSE` `S$=""` `WHILE X>0` `S$=CHR$(X MOD A+` `  ASC("0"))+S$` `X=X\A` `WEND` `ENDIF` `PRINT S$`

Заметим, что на языке Си такой метод реализовать сложнее из-за более сложной обработки строк. Потому на языке Си мы привели другой метод – складываем получающиеся остатки в целочисленный массив, а потом выводим их в обратном порядке. Переменная k хранит количество заполненных элементов в этом массиве, переменная i – счетчик цикла при выводе.

Задания для самостоятельного решения

2.10. Напишите программу, переводящую десятичное число в шестнадцатеричную систему счисления.

2.11. Напишите программу, переводящую число, записанное в системе счисления с основанием a ($a \leq 10$) и хранящееся в строке s, в десятичную систему счисления (в вид целого числа).

Рекомендация

Для решения задачи воспользуйтесь схемой Горнера – если есть число x и к этому числу нужно справа приписать еще одну цифру, то нужно умножить x на основание системы счисления и прибавить к результату эту цифру.

2.12. Напишите программу, переводящую число, записанное в 16-ричной системе счисления и хранящееся в строке s, в десятичную систему счисления (и записать его в целочисленную переменную).

Нахождение наименьшего простого делителя натурального числа

Напомним, делитель числа – это такое натуральное число, при делении на которое остаток от деления равен нулю.

Простое число имеет только два делителя – единицу и само число.

Числа, для которых это не выполняется (у них более двух делителей) называются составными.

Самое маленькое простое число – 2. Единица не является ни простым, ни составным числом.

Обозначим исходное натуральное число как x.

Если, начиная с числа 2, последовательно перебирать все целые числа и проверять делимость на них x, то первый найденный делитель будет простым.

Отдельная тонкость – до каких пор нужно продолжать перебирать числа в поисках делителя. Если число простое – то остановка произойдет только на самом числе. Однако, такой метод будет работать слишком долго.

Простое рассуждение подскажет нам, что достаточно искать делители до числа $x/2$ (число x не может делиться на что-нибудь больше $x/2$ и меньше x).

Более мудрое рассуждение подсказывает, что достаточно остановиться на числе "корень квадратный из x" (если не найдется ни одного делителя до корня из x, то не будет и ни одного после). Если ни одного делителя от 2 до корня из x не будет найдено – само число является простым:

На Паскале:	На Си:	На Бейсике:
`k:=2;` `while (k<=sqrt(x))and` `  (x mod k<>0) do` `  k:=k+1;` `if x mod k=0 then` `  write(k)` `else` `  write(x);`	`k=2;` `while(k<=sqrt(x) &&` `  x%k!=0)` `  k++;` `if(x%k==0)` `  printf("%d",k);` `else` `  printf("%d",x);`	`K=2` `WHILE K<=SQRT(X) AND` `  X MOD K<>0` `K=K+1` `WEND` `IF X MOD K=0 THEN` `PRINT K` `ELSE` `PRINT X` `ENDIF`

Заметим, что условие `k<=sqrt(x)` правильнее заменить на `k*k<=x`. Так мы избежим погрешностей вещественной арифметики, да и операция умножения выполняется быстрее, чем извлечение квадратного корня.

Задания для самостоятельного решения

2.13. Проверьте, является ли данное натуральное число простым.

Рекомендация

Найдите наименьший простой делитель числа, не превосходящий его квадратного корня. Если его нет – число простое.

2.14. Найдите все делители данного натурального числа.

Обработка массивов

Напомним, что такое массив и каковы основные операции работы с массивами.

Массив – совокупность однотипных данных, имеющих общее имя. Каждая такая единица данных называется *элементом массива*. Каждый элемент массива имеет уникальный номер, называемый *индексом* элемента массива.

В языках программирования обычно для указания нужного элемента массива указывают имя самого массива, рядом с которым, в скобках – индекс этого элемента. Например, для указания 5-го элемента массива MAS нужно написать MAS[5] (на Паскале и Си) или MAS(5) – на Бейсике.

Обратите внимание! Очень важно отличать значения элементов массива от индексов этих элементов. То, что в квадратных (на Бейсике – круглых) скобках – индекс! Если написано имя массива, квадратные (круглые) скобки и число/переменная/выражение внутри – это значение элемента.

Еще раз: в приведенном примере "5" – номер (индекс) элемента массива, MAS – имя массива, MAS[5] (MAS(5)) – значение 5-го элемента массива.

Часто (но не всегда) индекс элемента и значение элемента относятся к различным типам данных.

Так, например, индексом элемента массива может быть только значение перечислимого (порядкового) или ограниченного типа (в языке Си это вообще может быть только целое число).

Поэтому при написании программы будьте особенно внимательны, не путайте индексы элементов массива со значениями элементов массива.

Описание массива

Для использования в программе массив нужно обязательно описать. Это значит, что Вы предписываете компьютеру зарезервировать область памяти для хранения всех элементов вашего массива, указываете его имя, тип данных для элементов и размер (количество элементов).

Для всех трех языков мы будем описывать массив MAS, состоящий из 25 вещественных чисел.

На Паскале:	На Си:	На Бейсике:
В разделе описания переменных Var: `имя_массива: array[начальный_индекс.. конечный_индекс] of тип_элементов;` Пример, для элементов – вещественных чисел, пронумерованных от 1 до 25, нужно написать: `MAS: array[1..25]of real;` Заметим, что на Паксале (в отличие от языка Си, но как и в "обычном мире") принято элементы массива нумеровать начиная с первого. Заметим, что из трех официально поддерживаемых ЕГЭ языков программирования Паскаль предоставляет наибольшую гибкость в создании массивов с определенными индексами. Так, например, только в Паскале можно создать массив с индексами от 10 до 20 или от `'a'` до 'z' или даже указать в качестве индексов тип `char`. Впрочем, эти удобства, возможно, облегчат вам программирование некоторых задач С4. Для задач С2 можно всегда считать, что массивы нумеруются с единицы (или с нуля, если вам так удобнее).	В начале программы: `тип_данных имя_массива[количество_элементов];` Пример: `float MAS[25];` Напоминаем, что в языке Си номера элементов массива всегда начинаются с нуля. То есть, описание `MAS[n]` означает, что элемент с наименьшим индексом – `MAS[0]`, а элемент с наибольшим индексом – `MAS[n-1]`.	В начале программы: `DIM имя_массива(количество_элементов) AS тип_данных` Пример: `DIM MAS(25) AS INTEGER` Напоминаем, что на языке Бейсик при указании определенного числа в операторе `DIM`, в действительности выделяется на одну ячейку памяти больше, чем это число. Индексы получившихся элементов будут от нуля до этого числа. В приведенном примере создастся массив из 26-ти элементов, от `MAS(0)` до `MAS(25)`. Часто, для удобства (и большего понимания) про элемент с нулевым номером забывают и считают, что элементы нумеруются с первого.

При работе с массивами правильнее описать отдельную константу – количество элементов массива и в программе вместо числа 25 везде использовать именно ее. Это позволит быстро "изменять размерность задачи" – достаточно будет поменять одно число в начале программы и тогда вся программа начнет правильно работать при новом размере массива.

Это нужно написать примерно так:

На Паскале:	На Си:	На Бейсике:
`const n=25;` `var MAS: array[1..n] of real;`	`#define n 25` `float MAS[n];`	`N=25` `DIM MAS(N) AS REAL`

В наших примерах мы (для определенности) будем считать, что массив описан именно так – он называется `MAS` и в нем `n` элементов (`N + 1` в Бейсике).

Перечислим основные операции работы с массивами.

Прежде всего, вы должны понимать, что любая операция с массивами – это всегда обращение к нескольким (или всем) элементам с разными номерами. То есть, это почти всегда цикл. Если нужно обратиться ко всем элементам – то число повторений известно и наиболее подходящим является цикл FOR. Если нужно обратиться не ко всем элементам – это тоже цикл. Может быть, это будет не цикл FOR, но цикл – практически обязательно[2].

Если вы написали программу обработки массива в целом, и в ней нет цикла – значит, она неправильна, а вы (к сожалению) не понимаете, как обрабатываются массивы[3].

1. Задание начальных значений (инициализация) массива

Как правило, в задании С2 не требуется каким-либо способом задавать начальные значения элементов массива. И вы можете этого не делать, не опасаясь снижения оценки. Но в реальной программе значения массива, конечно, должны откуда-то браться. Поэтому мы, все же, рекомендуем написать две строчки ввода массива с клавиатуры. С нашей точки зрения, это создаст у эксперта более положительное впечатление о вашей компетентности – он будет знать, что вы понимаете про необходимость задания начальных значений. Тем более, что эти две строчки всегда одинаковы и при обучении работе с массивами они у вас уже (мы надеемся) доведены до автоматизма.

---

[2] Конечно, это замечание не вполне верно. Во многих языках однотипные массивы A и B можно присвоить одним оператором A:=B без явного цикла. Но если вам нужно проверить или изменить все элементы массива — нужно использовать цикл.

[3] Знатоки, конечно, здесь возмутятся. Ведь можно написать программу через, например, рекурсию. Она будет прекрасно обрабатывать массив, но при этом циклом не будет являться. Но рекурсия — это высокое искусство. Если вы умеете писать рекуррентные алгоритмы, то, вероятно, эту главу вам вообще лучше опустить :). К тому же, хвостовая рекурсия — то же самое, что итерация (цикл), только немного по другому организованная.

Напомним вам эти две строчки:

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n do` `  readln(MAS[i]);`	`for(i=0 ; i<n ; i++)` `  scanf("%f",&MAS[i]);`	`FOR I=1 TO N` `INPUT MAS(I)` `NEXT I`

2. Заполнение элементов массива определенными значениями

Чаще всего, если массив нужно заполнить начальными значениями, его нужно обнулить. Для этого нужно перебрать все элементы и в каждый положить ноль:

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n do` `  MAS[i]:=0;`	`for(i=0 ; i<n ; i++)` `  MAS[i]=0;`	`FOR I=1 TO N` `  MAS(I)=0` `NEXT I`

Другой, тоже очень распространенный случай – присвоить элементам массива значения их индексов:

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n do` `  MAS[i]:=i;`	`for(i=0 ; i<n ; i++)` `  MAS[i]=i;`	`FOR I=1 TO N` `  MAS(I)=I` `NEXT I`

Если найти формулу, используя которую, можно по индексу элемента вычислить его требуемое значение, то инициализация элементов массива – тоже простая задача.

Например, элементы массива должны принимать значения 1,4,7,10, …

Можно заметить, что это – арифметическая прогрессия. Первый элемент равен 1, каждый последующий на 3 больше предыдущего. То есть, разность прогрессии равна 3. Используем формулу i-го члена: $a_i=a_1+d(i-1)$. Подставляем $a_1=1$ и $d=3$, получаем $a_i=1+3(i-1)$. После раскрытия скобок и приведения подобных слагаемых получаем $a_i=3i-2$. Программа:

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n do` `  MAS[i]:=3*i-2;`	`for(i=0 ; i<n ; i++)` `  MAS[i]=3*i+1;`	`FOR I=1 TO N` `  MAS(I)=3*I-2` `NEXT I`

Для языка Си формула несколько другая (3i+1), т.к. в нем индексы нумеруются не с единицы, а с нуля.

Напоминаем, для каждой написанной вами программы обязательно делайте трассировку. В данном случае входные данные для программы отсутствуют, поэтому достаточно выполнить трассировку один раз. Нарисуйте на бумаге элементы массива (ряд последовательных клеток) и надпишите над каждой клеткой значение ее индекса. Теперь, присваивая переменной i (счетчику цикла) значения от начального и прибавляя по 1, вычисляйте значение выражения и помещайте его в клетки, номера которых равны i. Убедитесь, что массив заполняется правильно.

Даже если мы не можем (или не хотим) вывести формулу вычисления значения элемента массива по его индексу, но знаем, как вычислить значение элемента массива по его предыдущему элементу, можно воспользоваться таким способом:

Найдем правило, по которому вычисляется следующий элемент массива по предыдущему (в нашем случае прибавляется 3). Значит, в начальный элемент массива нужно положить требуемое число (1), а все остальные элементы вычислять, прибавляя число 3 к предыдущему:

На Паскале:	На Си:	На Бейсике:
`MAS[1]:=1;` `for i:=2 to n do` `  MAS[i]:=MAS[i-1]+3;`	`MAS[0]=1;` `for(i=1 ; i<n ; i++)` `  MAS[i]=MAS[i-1]+3;`	`MAS(1)=1` `FOR I=2 TO N` `  MAS(I)=MAS(I-1)+3` `NEXT I`

Сделайте трассировку программы, чтобы убедиться в правильности ее работы.

Более сложный случай – заполнить элементы массива последовательностью чисел Фибоначчи (1, 1, 2, 3, 5, 8, ...). Здесь правило – каждый последующий равен сумме двух предыдущих. Значит, достаточно задать значение двух начальных элементов массива, а для остальных указать, что они равны сумме двух предыдущих:

На Паскале:	На Си:	На Бейсике:
`MAS[1]:=1;` `MAS[2]:=1;` `for i:=3 to n do` `MAS[i]:=MAS[i-1]+` `  MAS[i-2];`	`MAS[0]=1;` `MAS[1]=1;` `for(i=2 ; i<n ; i++)` `  MAS[i]=MAS[i-1]+` `  MAS[i-2];`	`MAS(1)=1` `MAS(2)=1` `FOR I=3 TO N` `  MAS(I)=MAS(I-1)+MAS(I-2)` `NEXT I`

Сделайте трассировку программы, чтобы убедиться в правильности ее работы.

Задания для самостоятельного решения

При выполнении каждого задания обязательно выполните трассировку и убедитесь, что массив заполняется правильно!

2.15. Заполнить элементы массива последовательностью чисел: 2, 4, 6, 8, ...

2.16. Заполнить элементы массива последовательностью чисел: 2, 5, 8, 11, ...

2.17. Заполнить элементы массива последовательностью чисел: 2, 4, 8, 16, 32, ...

2.18. Заполнить элементы массива последовательностью чисел: 1, 3, 7, 15, 31, ...

3. Вычисление суммы всех элементов массива

Для этого нужно завести дополнительную переменную, положить в нее начальное значение – ноль. Потом перебрать все элементы и значение каждого добавить к этой переменной:

На Паскале:	На Си:	На Бейсике:
`sum:=0;` `for i:=1 to n do` `  sum:=sum+MAS[i];`	`sum=0;` `for(i=0 ; i<n ; i++)` `  sum+=MAS[i];`	`sum=0` `FOR I=1 TO N` `  sum=sum+MAS(I)` `NEXT I`

Еще раз подчеркнем значение трассировки программы. На экзамене у вас нет возможности запустить программу на компьютере или сверить ваше решение с ответом. Единственный способ проверить ее правильность – трассировка.

Не нужно думать, что "настоящие программисты обходятся без трассировки, поэтому и я этого делать не буду". Просто либо "настоящие программисты" имеют столько опыта, что способны быстро в голове "прокрутить" свою программу. Либо у них есть готовые примеры, на которых они свои программы проверяют (запускают). Либо с ними работают тестировщики – специалисты, которые специально только этим и занимаются. Либо эти "настоящие программисты" излишне самоуверенны и рискуют потерять работу.

Приучив себя делать трассировку каждый раз после написания программы, вы постепенно научитесь быстро находить ошибки и "прокручивать" трассировку в уме.

Сложность трассировки в этом и последующих примерах (в отличие от предыдущего) в том, что значения элементов массива заранее неизвестны. Потому наименьшее, что вы должны сделать – придумать хоть какой-нибудь вариант значений массива и отрассировать свою программу на нем. То есть, убедиться, что ваша программа работает хотя бы на одном примере входных данных. Это необходимо.

Задания для самостоятельного решения

2.19. Найти произведение всех элементов массива

2.20. Найти среднее арифметическое всех элементов массива

2.21. Найти среднее отличие элементов массива от их правого соседа

Пояснение

Для массива 5, 1, 3, 8 (из четырех элементов) имеется три пары соседних элементов (5 и 1, 1 и 3, 3 и 8). Отличия в парах составляют 4, 2, 5 соответственно. Среднее отличие равно (4+2+5)/3≈3,67).

4. Вычисление количества элементов массива, удовлетворяющих условию

Для этого нужно: завести дополнительную переменную (целочисленную) – счетчик количества таких элементов, присвоить ей начальное значение – ноль. Затем перебрать все элементы массива, значение каждого проверить на выполнение условия. Если условие выполняется – увеличить на единицу счетчик количества. Например, посчитаем количество четных элементов:

На Паскале:	На Си:	На Бейсике:
`kol:=0;` `for i:=1 to n do` `  if MAS[i] mod 2=0 then` `  kol:=kol+1;`	`kol=0;` `for(i=0 ; i<n ; i++)` `  if(MAS[i]%2 == 0)` `  kol++;`	`kol=0` `FOR I=1 TO N` `  IF MAS(I) MOD 2=0 THEN` `  kol=kol+1` `  ENDIF` `NEXT I`

Задания для самостоятельного решения

2.22. Найти количество отрицательных элементов массива

2.23. Найти среднее арифметическое всех нечетных элементов целочисленного массива

Замечание

Это тот случай, когда трассировки программы на одном варианте входных данных не достаточно. Потому что нужно проверить работу вашей программы на всех возможных различных вариантах входных данных. Под различными вариантами мы понимаем, конечно, не все-все различные числа, а все такие случаи, при которых про-

грамма должна вести себя по-разному. И чем сложнее задачу вы будете решать, тем большее количество случаев должна предусматривать ваша программа и тем на большем количестве тестов вы должны будете сделать трассировку. Научиться определять, какие случаи входных данных нужно предусмотреть и протестировать – это большое искусство. Это часть искусства программирования. Вероятнее всего, это приходит с опытом. В данном случае не забудьте рассмотреть случай, когда в массиве все элементы являются четными.

2.24. Найти количество всех элементов целочисленного массива, которые не являются положительными двузначными числами

5. Выполнение над элементами массива, удовлетворяющими условию, каких-то действий

Для этого нужно перебрать все элементы массива, для каждого проверить нужное условие. Если оно выполняется – выполнить требуемые действия. Иначе (если не выполняется) – выполнить другие действия.

Например, удвоить все положительные элементы массива, и поменять знак у всех остальных:

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n do` `  if MAS[i]>0 then` `  MAS[i]:=MAS[i]*2` `  else` `  MAS[i]:=-MAS[i];`	`for(i=0 ; i<n ; i++)` `  if(MAS[i]>0)` `  MAS[i]*=2;` `  else` `  MAS[i]=-MAS[i];`	`FOR I=1 TO N` ` IF MAS(I)>0 THEN` ` MAS(I)=MAS(I)*2` ` ELSE` ` MAS(I)=-MAS(I)` ` ENDIF` `NEXT I`

Задания для самостоятельного решения

2.25. Обнулить все отрицательные элементы массива и посчитать количество остальных.

2.26. Все четные положительные элементы целочисленного массива уменьшить вдвое, все нечетные положительные увеличить на 2, а у всех остальных поменять знак.

6. Перестановка всех элементов массива в обратном порядке

Задача решается путем перестановки элементов массива попарно – первый с последним, второй с предпоследним, и т.д. Два тонких момента, которые нужно учесть – что количество таких перестановок равно вовсе не количеству элементов массива (очень распространенная ошибка), а в два раза меньше. И формула, которая по номеру элемента i вычисляет номер элемента, с которым его нужно поменять местами – n+1–i (n–1–i на Си):

На Паскале:	На Си:	На Бейсике:
`for i:=1 to n div 2 do` `begin` `  tmp:=MAS[i];` `  MAS[i]:=MAS[n+1-i];` `  MAS[n+1-i]:=tmp` `end;`	`for(i=0 ; i<n/2 ; i++)` `{` `  tmp=MAS[i];` `  MAS[i]=MAS[n-1-i];` `  MAS[n-1-i]=tmp;` `}`	`FOR I=1 TO N \ 2` `TMP=MAS(I)` `MAS(I)=MAS(N+1-I)` `MAS(N+1-I)=TMP` `NEXT I`

Заметим, что в случае нечетного количества элементов массива средний элемент должен просто остаться на месте. Этот случай не рассмотрен отдельно, потому что он учтен при вычислении числа повторений (операция целочисленного деления "отбросит" дробную часть от деления на 2).

Так как на языке Си элементы массива нумеруются с нуля, а последний элемент массива имеет номер n–1, то формула для обмена i-го элемента несколько другая (n–1–i).

Задания для самостоятельного решения

2.27. Развернуть обе половинки массива в обратном порядке. Считать, что массив имеет четное число элементов.

Пояснение

Из массива (1, 2, 3, 4, 5, 6, 7, 8) нужно получить массив (4, 3, 2, 1, 8, 7, 6, 5).

2.28. Сдвинуть все элементы массива на одну позицию влево (циклически). Первый элемент должен оказаться на месте последнего.

2.29. Сдвинуть все элементы массива на одну позицию вправо (циклически). Последний элемент должен оказаться на месте первого.

7. Проверить, есть ли в массиве хотя бы один четный элемент

Это очень важная задача для понимания тонкостей работы с массивами. Нужно проверить, для всех ли элементов массива выполняется определенное условие. Типичная ошибка в такой задаче – перебирать все элементы, для каждого проверять условие и, если оно выполняется, выводить "Да", а если не выполняется – выводить "Нет". Такая программа выведет на экран много слов "Да" и "Нет" и не будет иметь ничего общего с правильной программой. Важно понимать, что ответ нужно вывести на экран только один раз и что пока не проверишь ВСЕ элементы массива, нельзя утверждать, что хотя бы один из них не удовлетворяет какому-то условию.

Самый понятный (и достаточно эффективный) способ – посчитать, для скольких элементов выполняется искомое свойство. В конце проверить, чему равно это количество. Если нулю – таких элементов в массиве нет. Если не нулю – они есть.

На Паскале:	На Си:	На Бейсике:
`kol:=0;` `for i:=1 to n do` `  if MAS[i] mod 2=0 then` `  kol:=kol+1;` `if kol=0 then` `  writeln('нет четных')` `else` `  writeln('есть четные');`	`kol=0;` `for(i=0 ; i<n ; i++)` `  if(MAS[i]%2 == 0)` `  kol++;` `if (kol==0)` `  printf("нет четных");` `else` `  printf("есть четные");`	`kol=0` `FOR I=1 TO N` `  IF MAS(I) MOD 2=0 THEN` `  kol=kol+1` `  ENDIF` `NEXT I` `IF kol=0 THEN` `  PRINT "нет четных"` `ELSE` `  PRINT "есть четные"` `ENDIF`

Задания для самостоятельного решения

2.30. Проверить, что в массиве все элементы положительны.

2.31. Проверить, что все элементы массива равны друг другу.

2.32. Проверить, есть ли в массиве элемент, равный заданному значению.

8. Проверка упорядоченности массива

Для определенности – по возрастанию (неубыванию). Введем дополнительную переменную-флажок (критерий упорядоченности). Присвоим ей начальное значение, равное нулю. Посчитаем, в каком количестве случаев порядок элементов в паре будет неверным. Для этого переберем все соседние пары элементов (их будет `n-1` штука). Если в паре левый элемент (с меньшим номером) оказался больше, чем правый, то увеличим на единицу переменную-флажок. После окончания цикла проверим, изменилась ли переменная-флажок. Если она осталась равной нулю – значит, во всех парах порядок верный и массив упорядочен. Иначе – массив неупорядочен:

На Паскале:	На Си:	На Бейсике:
`flag:=0;` `for i:=1 to n-1 do` `  if MAS[i]>MAS[i+1] then` `  flag:=flag+1;` `if flag=0 then` `  write('упорядочен')` `else` `  write('неупорядочен');`	`flag=0;` `for(i=0 ; i<n-1 ; i++)` `  if(MAS[i]>MAS[i+1])` `  flag++;` `if(flag==0)` `  printf("упорядочен");` `else` `  printf("неупорядочен");`	`flag=0` `FOR I=1 TO N-1` `  IF MAS(I)>MAS[I+1] THEN` `  flag=flag+1` `  ENDIF` `NEXT I` `IF flag=0 THEN` `  PRINT "упорядочен"` `ELSE` `  PRINT "неупорядочен"` `ENDIF`

Обратите внимание на номера сравниваемых элементов – мы перебираем в цикле элементы от начального до предпоследнего. Это значит, что мы должны текущий элемент сравнивать со следующим. Если счетчик цикла равен i (номер текущего элемента), то номер следующего элемента – на единицу больше. То есть, i+1.

Задания для самостоятельного решения

2.33. Проверить, что массив упорядочен строго по убыванию (каждый последующий элемент строго меньше предыдущего).

9. Поиск максимального элемента произвольного массива

Общая идея поиска максимума (минимума) такая: возьмем некоторое значение в качестве начального значения максимума. Затем переберем все элементы и проверим, есть ли элемент, который больше, чем этот выбранный максимум. Если такой нашелся – будем теперь считать его максимальным и сравним с остальными.

Отдельного внимания заслуживает выбор начального значения максимума.

Если про значения элементов массива ничего не известно, в качестве начального значения нужно выбирать первый элемент массива.

Итак, будем считать, что максимальный элемент – начальный. Проверим, так ли это. Положим его значение в переменную max. Переберем все остальные элементы и будем сравнивать каждый из них с переменной max. Если для какого-то элемента окажется, что он больше, чем max (текущий кандидат в максимальные), то будем теперь считать, что он – максимальный (положим его значение в max). После окончания цикла переменная max будет содержать в себе значение наибольшего элемента массива:

На Паскале:	На Си:	На Бейсике:
`max:=MAS[1];` `for i:=2 to n do` `  if MAS[i]>max then` `  max:=MAS[i];`	`max=MAS[0];` `for(i=1 ; i<n ; i++)` `  if(MAS[i]>max)` `  max=MAS[i];`	`max=MAS(1)` `FOR I=2 TO N` `  IF MAS(I)>max THEN` `  max=MAS(I)` `  ENDIF` `NEXT I`

Обратите внимание, цикл начинается не с начального элемента, а со следующего.

Очевидно, что если нужно найти наименьший элемент, достаточно поменять знак операции сравнения на "<". Хотя, для лучшего понимания программы, лучше еще поменять и имя переменной "max" на "min".

10. Поиск количества элементов произвольного массива, равных максимальному

Данная задача имеет два метода решения. Первый – сначала (за один проход по массиву) найти значение максимального элемента. Затем (за второй проход по массиву) сравнить все элементы с найденным максимумом и посчитать количество равных.

Второй способ – сделать это одновременно, за один проход по массиву.

Оба эти способа работают с одинаковой скоростью и считаются одинаково эффективными. Но второй считается "красивее". Его "красивость" состоит в том, что каждый элемент массива при этом просматривается только один раз. Таким образом, такой способ может быть применим не только к массиву, но и к задаче, в которой все просматриваемые значения не хранятся, а просто поступают откуда-то последовательно (например, вводятся с клавиатуры).

В этом случае однопроходный алгоритм позволяет не хранить все просмотренные значения – значит, экономит память.

Идея метода состоит в том, чтобы кроме значения максимума хранить сразу и их количество. Если очередной элемент оказывается лучше (больше) текущего максимума – меняем максимум и считаем, что таких максимумов встретилось к настоящему моменту один. Если же текущий элемент не больше максимума, сравниваем его с максимумом на равенство. Если равны – счетчик максимумов увеличиваем на 1:

На Паскале:	На Си:	На Бейсике:
`max:=MAS[1];` `nmax:=1;` `for i:=2 to n do` `  if MAS[i]>max then` `  begin` `  max:=MAS[i];` `  nmax:=1` `  end` `  else` `  if MAS[i]=max then` `  nmax:=nmax+1;`	`max=MAS[0];` `nmax=1;` `for(i=1 ; i<n ; i++)` `  if(MAS[i]>max)` `  {` `  max=MAS[i];` `  nmax=1;` `  }` `  else` `  if(MAS[i]==max)` `  nmax++;`	`max=MAS(1)` `nmax=1` `FOR I=2 TO N` `IF MAS(I)>max THEN` `max=MAS(I)` `nmax=1` `ELSE` `IF MAS(I)=max THEN` `nmax=nmax+1` `ENDIF` `ENDIF` `NEXT I`

Обратите внимание, что если слово else из программы убрать, то она будет работать неверно!

Задания для самостоятельного решения

2.34. Найти номер максимального элемента массива, если он единственный, или количество максимальных элементов, если их несколько.

11. Поиск второго по величине максимального элемента массива

Данная задача, как и предыдущая, имеет два метода решения. Первый – за два прохода по массиву (ищем номер максимума, затем ищем максимум среди остальных элементов). Заметим, что если на втором проходе искать максимум среди элементов, не равных первому максимуму, то результат будет не совсем тот – программа найдет вто-

рое по величине значение элемента массива, а не значение элемента, который в упорядоченном по возрастанию массиве стоял бы предпоследним (как требуется), хотя могут встречаться задачи, где будет требоваться именно это – найти два самых больших различных значения.

Нас будет (по тем же причинам, что и ранее) интересовать однопроходный алгоритм поиска второго максимума.

Идея метода такая – вводим две переменные (первый и второй максимумы) и задаем им начальные значения.

Затем перебираем все элементы массива и сравниваем их сначала с максимумом. Если текущий элемент лучше (больше) текущего максимума – значение максимума перекладываем в переменную второго максимума, а в максимум кладем значение текущего. Иначе – сравниваем текущий элемент со вторым максимумом. Если текущий больше – кладем его на место второго максимума.

Отдельного рассмотрения заслуживает задание начальных значений для обоих максимумов. Если про значения элементов массива заранее известно, что они не превышают некоего значения – самый просто случай – кладем в оба максимума меньшее значение. Если же про значения элементов ничего не известно – выбираем начальные значения из двух первых элементов массива. Большее положим в максимум, значение другого элемента – во второй максимум:

На Паскале:	На Си:	На Бейсике:
`max:=MAS[1];` `max2:=MAS[2];` `if max<max2 then` `begin` `  max:=MAS[2];` `  max2:=MAS[1]` `end;` `for i:=3 to n do` `  if MAS[i]>max then` `  begin` `  max2:=max;` `  max:=MAS[i]` `  end` `  else` `  if MAS[i]>max2 then` `  max2:=MAS[i];`	`max=MAS[0];` `max2=MAS[1];` `if(max<max2)` `{` `  max=MAS[1];` `  max2=MAS[0];` `}` `for(i=2 ; i<n ; i++)` `  if(MAS[i]>max)` `  {` `  max2=max;` `  max=MAS[i];` `  }` `  else` `  if(MAS[i]>max2)` `  max2=MAS[i];`	`max=MAS(1)` `max2=MAS(2)` `IF max<max2 THEN` `max=MAS(2)` `max2=MAS(1)` `ENDIF` `FOR I=3 TO N` `IF MAS(I)>max THEN` `max2=max` `max=MAS(I)` `ELSE` `IF MAS(I)>max2 THEN` `max2=MAS(I)` `ENDIF` `ENDIF` `NEXT I`

Задания для самостоятельного решения

2.35. Найти номер второго по величине максимального элемента массива.

2.36. Найти номер третьего по величине максимального элемента массива.

2.37. Найти второе по величине значение элемента массива. Возможно, в массиве все элементы одинаковы.

12. Поиск максимального элемента массива, про значения элементов которого известно, что они принадлежат определенному диапазону

Если заранее известно, что значения элементов массива ограничены определенными значениями (например, от –500 до +500), то в качестве начального значения можно взять значение, про которое заранее известно, что оно меньше любого элемента массива. В данном примере это может быть –501.

На Паскале:	На Си:	На Бейсике:
max:=-501; for i:=1 to n do if MAS[i]>max then max:=MAS[i];	max=-501; for(i=0 ; i<n ; i++) if(MAS[i]>max) max=MAS[i];	max=-501 FOR I=1 TO N IF MAS(I)>max THEN max=MAS(I) ENDIF NEXT I

Обратите внимание, теперь цикл начинается с начального элемента, чтобы не пропустить тот случай, когда он является максимальным элементом массива.

Задания для самостоятельного решения

2.38. Найти минимальный элемент массива, если известно, что значения элементов массива положительны и не превосходят 100.

2.39. Найти максимальный элемент массива, который больше 150. Известно, что значения элементов массива положительны и не превосходят 200.

13. Поиск номера максимального элемента

Хотя эта задача очень похожа на предыдущую задачу, пока она нами не решена. Так как, зная значение наибольшего элемента, мы не знаем, какой у него индекс. Либо теперь нужно еще раз пройти по массиву и сравнить каждый элемент с max, либо нужно переписать программу, чтобы она сразу искала индекс наибольшего.

Заметим, что если знать индекс наибольшего, то значение наибольшего элемента по нему получить очень просто.

Используем ту же технологию, но теперь в дополнительной переменной будем хранить не значение текущего максимума, а его номер. Обозначим эту переменную через imax. Ее начальное значение будет равно номеру начального элемента. Сравнивать теперь мы будем текущий элемент (с номером i) с текущим наибольшим элементом (с номером imax). При несоответствии – положим в imax значение текущего номера:

На Паскале:	На Си:	На Бейсике:
imax:=1; for i:=2 to n do if MAS[i]>MAS[imax] then imax:=i;	imax=0; for(i=1 ; i<n ; i++) if(MAS[i]>MAS[imax]) imax=i;	imax=1 FOR I=2 TO N IF MAS(I)>MAS(imax) THEN imax=I ENDIF NEXT I

Задания для самостоятельного решения

2.40. Найти номер минимального элемента массива, если известно, что значения элементов массива лежат в диапазоне 100..200.

2.41. Найти номер максимального элемента массива, который меньше 100. Известно, что значения элементов массива положительны и не превосходят 200.

14. Поиск максимального элемента массива, удовлетворяющего определенному условию (например, максимального отрицательного)

Особенностью этой задачи является то, что в качестве начального значения максимума нельзя принять первый элемент массива, так как совсем не обязательно, что он удовлетворяет условию (в данном случае – что он отрицательный).

Если про значения элементов массива известно, что они принадлежат определенному диапазону (например, от –500 до +500), то в качестве начального значения можно, как и в предыдущем примере, взять значение, которое гарантированно меньше любого элемента массива:

На Паскале:	На Си:	На Бейсике:
max:=-501; for i:=1 to n do if MAS[i]<0 then if MAS[i]>max then max:=MAS[i];	max=-501; for(i=0 ; i<n ; i++) if(MAS[i]<0) if(MAS[i]>max) max=MAS[i];	max=-501 FOR I=1 TO N IF MAS(I)<0 THEN IF MAS(I)>max THEN max=MAS(I) ENDIF ENDIF NEXT I

Если же про значения элементов массива ничего заранее неизвестно, задача существенно усложняется. Приходится сначала решать задачу поиска первого подходящего элемента (в нашем примере – первого отрицательного), а потом перебирать оставшиеся элементы (отрицательные) и сравнивать их с уже найденным элементом:

На Паскале:	На Си:	На Бейсике:
k:=1; while MAS[k]>=0 do k:=k+1; max:=MAS[k]; for i:=k+1 to n do if MAS[i]<0 then if MAS[i]>max then max:=MAS[i];	k=0; while(MAS[k]>=0) k++; max=MAS[k]; for(i=k+1 ; i<n ; i++) if(MAS[i]<0) if(MAS[i]>max) max=MAS[i];	k=1 WHILE MAS(k)>=0 DO K=K+1 WEND max=MAS(k) FOR I=k+1 TO N IF MAS(I)<0 THEN IF MAS(I)>max THEN max=MAS(I) ENDIF ENDIF NEXT I

Заметим, что в таком виде задачу можно решать, только если заранее известно, что в массиве есть хотя бы один элемент, удовлетворяющий условию (в нашем примере, отрицательный). Если это не так, нужно добавить еще одно условие, которое нужно чтобы не выйти в первом цикле за границу массива:

На Паскале:	На Си:	На Бейсике:
k:=1; while (k<n) and (MAS[k]>=0) do k:=k+1; if MAS[k]<0 then begin max:=MAS[k]; for i:=k+1 to n do if MAS[i]<0 then if MAS[i]>max then max:=MAS[i]; write(max) end else write('таких нет');	k=0; while(k<n-1 && MAS[k]>=0) k++; if(MAS[k]<0) { max=MAS[k]; for(i=k+1 ; i<n ; i++) if(MAS[i]<0) if(MAS[i]>max) max=MAS[i]; printf("%f",max); } else printf("таких нет");	k=1 WHILE k<n AND MAS(k)>=0 DO K=K+1 WEND IF MAS(k)<0 THEN max=MAS(k) FOR I=k+1 TO N IF MAS(I)<0 THEN IF MAS(I)>max THEN max=MAS(I) ENDIF ENDIF PRINT max NEXT I ELSE PRINT "таких нет" ENDIF

Существует более элегантный (хотя и не такой понятный) способ решения этой задачи:

На Паскале:	На Си:	На Бейсике:
k:=0; for i:=1 to n do if MAS[i]<0 then if (k=0) or (MAS[i]>MAS[k]) then k:=i; if k>0 then write(MAS[k]) else write('таких нет');	k=0; for(i=0 ; i<n ; i++) if(MAS[i]<0) if(k==0 || MAS[i]>MAS[k]) k=i; if(k>0) printf("%f",MAS[k]); else printf("таких нет");	k=0 FOR I=1 TO N IF MAS(I)<0 THEN IF k=0 OR MAS(I)>MAS(k) THEN k=I ENDIF ENDIF NEXT I IF k>0 THEN PRINT MAS(k) ELSE PRINT "таких нет" ENDIF

Задания для самостоятельного решения

2.42. Найти минимальный четный элемент целочисленного массива, если известно, что значения элементов массива лежат в диапазоне –200..+200, и в массиве есть хотя бы один четный элемент.

2.43. Найти минимальный четный элемент целочисленного массива.

2.44. Найти номер минимального двузначного элемента целочисленного массива, если известно, что значения элементов массива лежат в диапазоне 1..200, и в массиве есть хотя бы один двузначный элемент.

2.45. Найти номер минимального двузначного элемента целочисленного массива.

15. Поиск номера элемента, удовлетворяющего определенному условию

Самый простой случай – поиск номера элемента, равного указанному значению (x). При этом нужно учитывать три случая – в массиве ровно один такой элемент, в массиве несколько таких элементов, либо в массиве нет указанного элемента. Если ничего в условии не сказано, при наличии нескольких нужных элементов, достаточно указать любой. Например, первый по счету. При этом, если специально не указано, можно этот случай не рассматривать отдельно, а просто найти первый нужный элемент.

А вот случай, когда нужного элемента нет, нужно обязательно рассмотреть отдельно.

Самый "правильный" способ решения задачи – перебирать подряд все элементы массива, начиная с первого, пока не будет выполнено нужное условие или пока не кончится массив. Если массив кончится, а условие ни разу не выполнится – вывести сообщение, что нужного элемента нет.

Такой метод решения предполагает цикл с двумя условиями и обязательную проверку, по какому из условий закончился цикл:

На Паскале:	На Си:	На Бейсике:
i:=1; while (i<n) and (MAS[i]<>x) do i:=i+1; if MAS[i]=x then write(i) else write('таких нет');	i=0; while(i<n-1 && MAS[i]!=x) i++; if(MAS[i]==x) printf("%d",i); else printf("таких нет");	I=1 WHILE I<N AND MAS(I)<>X I = I + 1 WEND IF MAS(I)=X THEN PRINT I ELSE PRINT "таких нет" ENDIF

Обратите внимание, в цикле проверяется не условие совпадения (что текущий элемент равен *x*), а условие несовпадения – ведь увеличение счетчика на 1 должно произойти, если мы пока НЕ НАШЛИ нужный элемент.

Еще один важный момент – цикл продолжается не до N-го элемента массива, а до N–1-го. Это нужно предусмотреть для случая, когда в массиве нет нужного элемента. Ведь если поставить условие (i<=n) AND (MAS[i]<>x), то оба они на этом шаге тоже выполнятся, значение переменной i увеличится на 1 и будет проверяться это условие для i=n+1. Для языка Си это не страшно. А вот для других языков, в зависимости от компилятора, это может привести к аварийному завершению программы – ведь проверяется условие MAS[i]<>x для i=n+1. А элемента массива с индексом n+1 нет! Это ошибка времени выполнения – "выход за границу массива".

Чтобы этого не произошло, мы повторяем цикл не далее n–1-го элемента.

Чтобы после окончания цикла определить, есть ли в массиве нужный нам элемент, мы проверим значение элемента массива MAS[i]. Если цикл остановился при нахождении нужного нам элемента – проверка выполнится и программа выведет искомый номер – i. Если цикл остановился по достижении конца массива – проверится его последний элемент. Если он тот, кто нам нужен, проверка, опять же, выполнится и программа выведет нужный номер. Если проверка не выполнится, значит и последний элемент массива – не тот, кто нам нужен и выведется сообщение об отсутствии таких элементов.

Задания для самостоятельного решения

2.46. Найти номер второго по счету элемента массива, равного *x*. Если в массиве меньше двух таких элементов – вывести об этом сообщение.

2.47. Найти номер первого по счету нечетного элемента массива. Известно, что в массиве есть хотя бы один нечетный элемент.

2.48. Найти номер первого по счету нечетного элемента массива.

2.49. Найти номер последнего по счету положительного элемента массива. Известно, что в массиве есть хотя бы один положительный элемент.

2.50. Найти номер последнего по счету положительного элемента массива.

16. Слияние двух упорядоченных массивов

Из двух массивов (например, MAS1 и MAS2), каждый из которых упорядочен (например, по неубыванию), получить новый массив (например, MAS3), состоящий из элементов обоих исходных массивов, таким же образом упорядоченный.

Первое, что приходит в голову – скопировать в результирующий массив сначала все элементы первого массива, затем все элементы второго массива, после чего отсортировать полученный массив. Хотя этот алгоритм и приведет нас к решению задачи, он очень неэффективен – долго работает по сравнению с оптимальным решением.

Для эффективного решения нужно воспользоваться тем, что элементы каждого массива уже упорядочены. Введем две переменных-счетчика (i и j), каждый из которых изначально "установим" на начало первого и второго массивов соответственно. Сравним между собой элементы массивов, на которые "указывают" счетчики. Меньший из них запишем в результирующий массив и "передвинем" на следующий элемент счетчик того массива, откуда был взят элемент.

Так будем продолжать делать до тех пор, пока один из массивов не "закончится". После этого оставшуюся часть другого массива допишем в конец результирующего массива.

На Паскале:	На Си:	На Бейсике:
const n=25; m=30; var MAS1:array[1..n]of real; MAS2:array[1..m]of real; MAS3:array[1..n+m] of real; i,j,k:integer; ... i:=1; j:=1; while (i<=n)and(j<=m)do if MAS1[i]<MAS2[j] then begin MAS3[i+j-1]:=MAS1[i]; i:=i+1 end else begin MAS3[i+j-1]:=MAS2[j]; j:=j+1 end; if i<=n then for k:=i to n do MAS3[m+k]:=MAS1[k] else for k:=j to m do MAS3[n+k]:=MAS2[k];	#define n 25 #define m 30 float MAS1[n],MAS2[m]; float MAS3[n+m]; int i,j,k; ... i=0; j=0; while(i<n && j<m) if(MAS1[i]<MAS2[j]) { MAS3[i+j]=MAS1[i]; i++; } else { MAS3[i+j]=MAS2[j]; j++; } if(i<n) for(k=i ; k<n ; k++) MAS3[m+k]=MAS1[k]; else for(k=j ; k<m ; k++) MAS3[n+k]=MAS2[k];	N=25 M=30 DIM MAS1(N),MAS2(M) DIM MAS3(N+M) ... I=1 J=1 WHILE I<=N AND J<=M IF MAS1(I)<MAS2(J) THEN MAS3(I+J-1)=MAS1(I) I=I+1 ELSE MAS3(I+J-1)=MAS2(J) J=J+1 ENDIF WEND IF I<=N THEN FOR K=I TO N MAS3(M+K)=MAS1(K) NEXT K ELSE FOR K=J TO M MAS3(N+K)=MAS2(K) NEXT K ENDIF

17. Сортировка массива

Задача сортировки массива (упорядочивания его элементов в порядке возрастания/убывания) может быть решена массой различных способов.

Однако на экзамене вам достаточно владеть любым способом, даже самым простым и неэффективным. Поэтому мы не будем рассматривать быстрые сортировки, а остановимся на самой простой.

Для удобства будем упорядочивать массив по возрастанию. Это значит, что в отсортированном массиве никакой элемент не должен быть меньше своего соседа слева. Конечно, правильнее было бы говорить о "неубывании". Но обычно это подразумевают, и используют термин "возрастание".

Метод, который традиционно считается самым простым, "базовым", называется сортировка "пузырьком".

Суть метода состоит в том, что последовательно сравниваются все пары соседних элементов массива и, если значения элементов в паре стоят в неправильном порядке (правый меньше левого), то они меняются местами.

В результате одного такого прохода по массиву самый большой элемент обязательно окажется на последней позиции массива. При этом от своей позиции он, путем постепенных обменов, дойдет до последнего места. "Всплывет" как пузырек.

Но массив при этом вовсе не обязательно окажется упорядоченным. Гарантированно на нужном месте при этом окажется только самый большой элемент. Поэтому эту процедуру нужно повторить. Но только теперь можно не рассматривать самую последнюю пару.

В результате предпоследний элемент тоже окажется на нужном месте.

Значит, всю процедуру нужно повторить столько раз, сколько элементов нужно поставить на нужное место. То есть, n–1 (оставшийся элемент окажется при этом на нужном месте сам):

На Паскале:	На Си:	На Бейсике:
`for k:=n-1 downto 1 do` `for i:=1 to k do` `if MAS[i]>MAS[i+1] then` `begin` `tmp:=MAS[i];` `MAS[i]:=MAS[i+1];` `MAS[i+1]:=tmp` `end;`	`for(k=n-2 ; k>=0 ; k--)` `for(i=0 ; i<k ; i++)` `if(MAS[i]>MAS[i+1])` `{` `tmp=MAS[i];` `MAS[i]=MAS[i+1];` `MAS[i+1]=tmp;` `}`	`FOR K=N-1 TO 1 STEP -1` `FOR I=1 TO K` `IF MAS(I)>MAS(I+1) THEN` `TMP=MAS(I)` `MAS(I)=MAS(I+1)` `MAS(I+1)=TMP` `ENDIF` `NEXT I` `NEXT K`

Задания для самостоятельного решения

2.51. Отсортировать массив по убыванию.

2.52. Отсортировать массив по возрастанию, просматривая при этом пары элементов массива справа налево.

Обработка строк

Обработка строк на разных языках программирования реализована по-разному.

Строки могут рассматриваться как массивы символов, к каждому символу которой можно обратиться как к элементу массива. Или как специальные объекты-строки, к которым применимы специальные строковые операции.

Так, например, в Паскале со строковыми переменными можно легко работать и как с массивами символов и как со специальными строковыми объектами.

В языке Си строка – это, прежде всего, массив символов. Чтобы работать с ней как с объектом-строкой, нужно подключать специальную библиотеку (и при этом все равно всегда помнить о том, как строка хранится и каковы особенности работы с ней).

В языке Бейсик со строкой нельзя работать как с массивом. Это, в частности, создает некоторые трудности при простом переборе символов строки или при замене символов строки на другие.

Напомним основные действия со строками.

На Паскале:	На Си:	На Бейсике:
Описание `s:string;` Метод хранения `В нулевом символе строки хранится длина строки` Копирование строки d в строку s `s:=d;` Добавление строки d к строке s `s:=s+d;` Обращение к символу строки с номером k как к элементу массива `s[k]` Вычисление длины строки `length(s)` Извлечение (копирование) из строки s символов, начиная с позиции n, k штук `copy(s,n,k)` Вставка в строку s символов строки d, начиная с позиции n `Insert(d,s,n);` Удаление из строки s символов, начиная с позиции n, k штук `Delete(s,n,k);` Поиск в строке s вхождения строки d. Если строка d будет найдена – номер позиции первого вхождения. Если не найдена – 0 `Pos(d,s)`	Описание `char s[256];` Метод хранения `Строка заканчивается символом с кодом 0` Копирование строки d в строку s `strcpy(s,d);` Добавление строки d к строке s `strcat(s,d);` Обращение к символу строки с номером k как к элементу массива `s[k]` Вычисление длины строки `strlen(s)`	Описание `DIM S AS STRING` Использование `К имени строковой переменной обязательно нужно добавлять символ доллара: S$` Копирование строки D$ в строку S$ `S$=D$` Добавление строки D$ к строке S$ `S$=S$+D$` Обращение к символу строки S$ с номером K `MID$(S$,K,1)` Вычисление длины строки S$ `LEN(S$)` Извлечение (копирование) из строки S$ символов, начиная с позиции N, K штук `MID$(S$,N,K)` Извлечение (копирование) левой части строки S$ (K символов) `LEFT$(S$,K)` Извлечение (копирование) правой части строки S$ (K символов) `RIGHT$(S$,K)`

Подсчет частоты появления символа в строке

Будем считать, что в строке S ищется символ x. Переберем все символы строки, сравним каждый из них с x и посчитаем количество совпавших:

На Паскале:	На Си:	На Бейсике:
`k:=0;` `for i:=1 to length(s) do` `  if s[i]=x then` `  k:=k+1;` `write(k);` Другой вариант: `k:=0;` `for i:=1 to length(s) do` `  if copy(s,i,1)=x then` `  k:=k+1;` `write(k);`	`k=0;` `for(i=0; s[i]!='\0'; i++)` `  if(s[i]==x)` `  k++;` `printf("%d",k);`	`K=0` `FOR I=1 TO LEN(S$)` `IF MID$(S$,I,1)=X THEN` `K=K+1` `ENDIF` `NEXT I` `PRINT K`

Задания для самостоятельного решения

2.53. Посчитать количество вхождений строки "123" в исходную строку.

2.54. Посчитать количество слов в исходной строке. Слова считать отделенными друг от друга одним пробелом. Считать, что в начале и в конце строки пробелов нет.

2.55. Посчитать количество слов в исходной строке. Слова считать отделенными друг от друга одним или несколькими пробелами.

Поиск подстроки внутри данной строки.
Замена найденной подстроки на другую строку

Обозначим исходную строку *s*, искомую подстроку *x*, а другую строку – *d*.

На Паскале с использованием функции Pos задача решается очень просто: функцией Pos(x,s) получим положение подстроки *x* в строке *s*, удалим ее процедурой Delete и вставим на эту позицию строку *d* процедурой Insert.

На Си ввиду отсутствия возможности вычленить часть строки и сравнить его с искомой подстрокой придется использовать вложенные циклы. Будем перебирать все возможные положения, где в строке *s* может находиться строка *x* и сравнивать последовательно символы части строки *s* с символами строки *x*. Пока не найдем совпадение по всем символам или строка *s* не закончится.

На Бейсике будем извлекать из строки S$ последовательно, начиная с первой позиции, части строки S$ длины LEN(X$) и сравнивать со строкой X$. При совпадении – закончим цикл.

На Паскале можно реализовать алгоритм тем же способом, что и на Бейсике, но использование функции Pos проще:

На Паскале:

```
k:=Pos(x,s);
if k=0 then
  write('нет совпадений')
else
begin
  Delete(s,k,length(x));
  Insert(d,s,k)
end;
```

На Си:

```
k=0;
do
{
  flag=0;
  for(i=0 ;
      i<strlen(x) &&
      flag==0 ;
      i++)
    if(s[k+i]!=x[i])
      flag=1;
  if(flag==1)
    k++;
} while(flag==1 &&
  k<strlen(s)-strlen(x));
if(flag==0)
{
 if(strlen(x)!=strlen(d))
  if(strlen(x)>strlen(d))
   for(i=0 ;
       i<strlen(s)-
         strlen(x)-k+1;
       i++)
     s[k+strlen(d)+i]=
       s[k+strlen(x)+i];
  else
```

На Бейсике:

```
K=1
L=LEN(X$)
WHILE MID$(S$,K,L)<>X$
      AND K<LEN(S$)-L
K=K+1
WEND
IF MID$(S$,K,L)<>X$ THEN
PRINT "нет совпадений"
ELSE
S$=LEFT$(S$,K-1)+D$+
  RIGHT$(S$,
        LEN(S$)-(K-1+L))
ENDIF
```

На Паскале:	На Си:	На Бейсике:
	for(i=strlen(s)- strlen(x)-k; i>=0 ; i--) s[k+strlen(d)+i]= s[k+strlen(x)+i]; for(i=0 ; i<strlen(d) ; i++) s[k+i]=d[i]; } else printf("нет совпадений");	

Заметим, что на языке Си для замены найденной подстроки на другую строку прошлось анализировать, какова разница в длинах строк *x* и *d* и сдвигать символы строки *s* на эту разницу (циклом). После чего осталось положить на выделенное место строку *d* (циклом).

Задания для самостоятельного решения

2.56. Найти количество вхождений подстроки в строке.

2.57. Найти позицию последнего вхождения подстроки в строке.

2.58. Удалить все лишние пробелы в строке (оставить между словами ровно по одному пробелу).

2.59. Переставить в строке все слова в обратном порядке (например, вместо "мама мыла раму" должно стать "раму мыла мама"). Слова считать разделенными ровно одним пробелом. Считать, что в начале и конце строки пробелов нет.

2.60. Переставить в строке все слова в обратном порядке (например, вместо "мама мыла раму" должно стать "раму мыла мама"). Слова считать разделенными одним или несколькими пробелами. В результирующей строке между словами не обязательно оставлять столько же пробелов, сколько было в исходной.

ОПРЕДЕЛЕНИЕ ВЫИГРЫШНОЙ СТРАТЕГИИ ИГРЫ (АНАЛИЗ И ПОСТРОЕНИЕ ДЕРЕВА ИГРЫ)

Характеристика задания с развернутым ответом С3 ЕГЭ по информатике

В соответствии со спецификацией экзамена, задание С3 нацелено на проверку умения построить дерево игры по заданному алгоритму и обосновать выигрышную стратегию. В отличие от других заданий части С, задание С3 не требует знания какого-либо языка программирования, поэтому оно часто выбирается экзаменуемыми для выполнения как простое. Однако, при кажущейся простоте, очень часто это задание выполняется неверно, и экзаменуемые не получают за него ни одного балла из 3-х возможных.

Рассмотрим, за что в данном задании предусмотрены три балла.

Максимальный балл (3) за выполнение задания предусмотрен, если верно указан игрок-победитель, приведена и строго обоснована его выигрышная стратегия. Два балла дается, если обоснование стратегии неполно или содержит ошибки. Один балл – если сама стратегия описана лишь частично.

Итак, для получения желаемых трех баллов нужно:

1) указать выигрывающего игрока,

2) указать стратегию его игры (указать все варианты хода его противника и соответствующие им ответы),

3) привести доказательство того, что приведенная стратегия игры – правильная.

Типичное условие и критерии оценивания задания С3, их анализ

Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (–3,2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+5,y), или в точку с координатами (x,y+4), или в точку с координатами (x+3,y+3). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 12 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

Самая главная ошибка, которую допускают экзаменуемые при выполнении задания С3 (как, впрочем, и при выполнении всех остальных заданий) – приходят на экзамен без подготовки, не попытавшись решить хотя бы демо-версию экзамена и не сверив свое решение с правильным ответом.

Увидев в первый раз такую задачу, экзаменуемые просто отвечают на первые два поставленные вопроса и полагают, что задача при этом практически решена. Наиболее

типичный ответ выглядит так: "Выиграет первый игрок. Он должен передвинуть фишку вправо". Если заглянуть в критерии оценивания (мы их приведем ниже), за такой ответ ставится 0 (ноль) баллов.

Оставим для другого случая обсуждение того, что в задаче спрашивается почти только это, и что экзаменуемый ответил на два вопроса из трех. Наша задача – не искать "зацепки" в тексте условия, а научиться решать задачу на максимальный балл.

Для этого прежде всего нужно понять, что же от нас хотят.

Лучше всего это можно сделать, анализируя критерии оценивания задания:

Указания по оцениванию	Баллы
Правильное указание выигрывающего игрока и его ходов со строгим доказательством правильности (с помощью или без помощи дерева игры).	**3**
Правильное указание выигрывающего игрока, стратегии игры, приводящей к победе, но при отсутствии доказательства ее правильности.	**2**
При наличии в представленном решении одного из пунктов: **1. Правильно указаны все варианты хода первого игрока и возможные ответы второго игрока (в том числе и все выигрышные), но неверно определены дальнейшие действия и неправильно указан победитель.** **2. Правильно указан выигрывающий игрок, но описание выигрышной стратегии неполно и рассмотрены несколько (больше одного, но не все) вариантов хода первого игрока и частные случаи ответов второго игрока.**	**1**
Задание не выполнено или в представленном решении полностью отсутствует описание элементов выигрышной стратегии, и отсутствует анализ вариантов первого-второго ходов играющих (даже при наличии правильного указания выигрывающего игрока).	**0**
Максимальный балл	**3**

Проанализировав условие и критерии оценивания, сформулируем иначе вопросы, на которые мы должны ответить для полного решения поставленной задачи:

1) Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход?

2) Если выигрывает первый игрок, укажите его первый ход, а затем как он должен ходить при всех возможных ответах второго игрока. Для каждого из приведенных ответов первого игрока укажите все возможные ходы второго игрока и на них также укажите выигрышные ответы первого игрока. (Как правило, после этого первый игрок выигрывает).

3) Если выигрывает второй игрок, укажите все возможные первые ходы первого игрока и для каждого укажите выигрышные ходы второго игрока. Для каждого из них укажите все возможные ходы первого игрока и для каждого из них – выигрышные ходы второго игрока. (Как правило, после этого второй игрок выигрывает).

2–3) Другими словами, приведите стратегию выигрыша – на каждый возможный ход проигрывающего игрока нужно привести ответ выигрывающего.

4) Приведите доказательство (ряд умозаключений), на основании которых Вы сделали вывод, что ваше решение верно.

Технология выполнения задания С3

Существует несколько способов выполнения задания С3.

Сначала рассмотрим первый способ – методом построения полного дерева игры и определения по нему выигрывающего игрока.

С одной стороны, именно этот метод предполагается составителями данной задачи.

С другой стороны, он самый кропотливый и трудоемкий.

Построение полного дерева игры – это рассмотрение для каждой позиции игры всех возможных ходов. Затем для получившихся позиций – снова рассмотрение всех возможных ходов. И так до тех пор, пока по всем получившимся "направлениям" мы не достигнем конца игры (в рассмотренном примере – расстояние от точки (0,0) до текущей позиции фишки больше 12 единиц).

Если изобразить такое решение графически: от начальной позиции нарисовать три линии – возможные ходы – в следующие позиции, из каждой из них – по три линии в следующие позиции, и т.д., то такое изображение будет напоминать дерево с ветвями-линиями (если, конечно, рисовать начальную позицию в нижней части листа, а линии – вверх). Это и называется "деревом игры", которое упоминается в критериях оценивания.

Обычно дерево рисуют сверху вниз или слева направо.

Данное решение выглядит очень трудоемким, потому что для каждой позиции игры нужно указать три возможных варианта следующего хода (в других задачах С3 – четыре варианта). То есть, рассматривая каждый следующий ход, размер нашего дерева игры увеличивается в три раза.

Так как это учебная задача, числа в ней подобраны так, чтобы дерево, все же, имело разумные размеры. Но все равно, выигрыш редко достигается раньше четвертого хода. А это значит, что в дереве на 4-м ходу рассматриваются 3^4=81 вариант игры, правда, часто многие из них повторяются.

Более короткие методы решения задачи мы рассмотрим позже. Важно, чтобы Вы, прежде всего, понимали, как построить дерево игры и какие выводы нужно из него сделать. А уже потом можно будет думать об убыстрении решения.

Как мы уже писали, дерево игры обычно строят-рисуют сверху вниз или слева направо. Мы будет изображать его слева направо из-за способа представления данных (в книге). Ведь в дереве будет в результате около 81 позиции на 4-м уровне. Если изображать дерево сверху вниз, ширины страницы при этом не хватит. А вот в высоту 81 строчка вполне поместится.

Сначала рассмотрим все возможные варианты первого хода (их три):

Начальное положение	Первый ход I-го игрока
–3,2	2,2
	–3,6
	0,5

Затем для каждого положения после первого хода рассмотрим все возможные варианты второго хода:

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока
–3,2	2,2	7,2
		2,6
		5,5
	–3,6	2,6
		–3,10
		0,9
	0,5	5,5
		0,9
		3,8

Прежде чем продолжить рассматривать дальнейшие возможные ходы, пора подумать, когда же мы собираемся останавливаться и что считать критерием остановки. Хотя, конечно, правильнее было бы подумать об этом до построения дерева игры. Но нам хотелось, чтобы Вы сами вовремя осознали необходимость такого действия.

Здесь есть два пути – либо для каждой позиции дерева вычислять расстояние до точки (0,0) (проще – квадрат расстояния), по теореме Пифагора. Либо вычислить "граничные" целочисленные координаты, "после которых" игра заканчивается. Проще – первое, быстрее – второе. Пока, для простоты, воспользуемся первым способом. Допишем к нашему дереву для каждой позиции квадрат расстояния до точки (0,0):

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)
		2,6 (40)
		5,5 (50)
	–3,6 (45)	2,6 (40)
		–3,10 (109)
		0,9 (81)
	0,5 (25)	5,5 (50)
		0,9 (81)
		3,8 (100)

Напоминаем, игра заканчивается по достижении расстояния, большего 12-ти. Это значит, что квадрат расстояния должен быть *больше* 144. Пока что ни одна из рассмотренных позиций не превысила 144, поэтому для каждого получившегося положения снова рассматриваем все три возможных варианта хода:

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)	**12,2 (148)**
			7,6 (85)
			10,5 (125)
		2,6 (40)	7,6 (85)
			2,10 (104)
			5,9 (106)
		5,5 (50)	10,5 (125)
			5,9 (106)
			8,8 (128)
	–3,6 (45)	2,6 (40)	7,6 (85)
			2,10 (104)
			5,9 (106)
		–3,10 (109)	2,10 (104)
			–3,14 (205)
			0,13 (169)
		0,9 (81)	5,9 (106)
			0,13 (169)
			3,12 (153)
	0,5 (25)	5,5 (50)	10,5 (125)
			5,9 (106)
			8,8 (128)
		0,9 (81)	5,9 (106)
			0,13 (169)
			3,12 (153)
		3,8 (73)	8,8 (128)
			3,12 (153)
			6,11 (157)

Жирным выделены те позиции фишки, расстояние до которых от точки (0,0) больше 12. То есть, это выигрыш одного из игроков (в данном случае, первого). Ходы из этих позиций дальше можно не рассматривать – игра на этом закончится. Рассмотрим остальные возможные позиции игры (после второго хода второго игрока):

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)	**12,2 (148)**	–
			7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
		2,6 (40)	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		5,5 (50)	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
	–3,6 (45)	2,6 (40)	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		–3,10 (109)	2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			–3,14 (205)	–
			0,13 (169)	–
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
	0,5 (25)	5,5 (50)	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
		3,8 (73)	8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
			3,12 (153)	–
			6,11 (157)	–

Дерево игры в виде таблицы получилось, как мы и предупреждали, немаленькое, но зато мы получили результат – полное дерево игры. Видно, что в последнем столбце квадрат расстояния (в скобках) всегда больше 144, поэтому дальше игра во всех случаях продолжаться не будет.

Мы получили дерево игры, но пока что еще не получили ответа ни на один вопрос, который у нас спрашивают – кто выигрывает, как он при этом должен ходить и почему это так. Ответы на эти вопросы мы получим из анализа дерева игры:

Позиции, выделенные жирным шрифтом в таблице – выигрыш. То есть, если игрок в нее перейдет, то он сразу выиграет.

Значит, предыдущая позиция (из которой игрок ходил) является для него выигрышной (из этой позиции игрок может пойти так, чтобы выиграть). Обозначим эти позиции подчеркиванием (мы приводим измененную таблицу, при решении, конечно, нужно просто подчеркнуть позиции в имеющейся):

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)	**12,2 (148)**	–
			7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
		2,6 (40)	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		5,5 (50)	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
	–3,6 (45)	2,6 (40)	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		–3,10 (109)	2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			–3,14 (205)	–
			0,13 (169)	–
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
	0,5 (25)	5,5 (50)	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
		3,8 (73)	8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
			3,12 (153)	–
			6,11 (157)	–

Повторимся – мы обозначили как выигрышные (подчеркнутые) все те позиции, справа от которых есть выигрыш (выделено жирным). То есть, все те позиции, из которых можно одним ходом выиграть.

Заметим, что для выделения этих позиций мы "двигаемся по дереву игры" с конца – для каждой конечной "жирной" позиции выделяем ее предшественника как выигрышную позицию.

Теперь осуществим следующий шаг – определим проигрышные позиции.

Это те позиции, из которых куда бы ни пошел игрок, он обязательно попадет в выигрышную (для соперника) позицию, тем самым его соперник тут же выиграет.

Значит, это те позиции, из которых все три хода приводят в выигрышную позицию.

То есть, клетки, справа от которых все три клетки подчеркнуты. Таких позиций будет всего 4 во всей таблице – два раза позиция (2,6) и два раза позиция (5,5). Обозначим их жирным курсивом. Чтобы не загромождать наше рассуждение излишней огромной таблицей, мы опустили последний столбец (второй ход II-го игрока), который сейчас уже не важен, так как про все позиции второго хода I-го игрока мы знаем, какие они:

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)	**12,2 (148)**
			7,6 (85)
			10,5 (125)
		2,6 (40)	7,6 (85)
			2,10 (104)
			5,9 (106)
		5,5 (50)	10,5 (125)
			5,9 (106)
			8,8 (128)
	–3,6 (45)	***2,6 (40)***	7,6 (85)
			2,10 (104)
			5,9 (106)
		–3,10 (109)	2,10 (104)
			–3,14 (205)
			0,13 (169)
		0,9 (81)	5,9 (106)
			0,13 (169)
			3,12 (153)
	0,5 (25)	***5,5 (50)***	10,5 (125)
			5,9 (106)
			8,8 (128)
		0,9 (81)	5,9 (106)
			0,13 (169)
			3,12 (153)
		3,8 (73)	8,8 (128)
			3,12 (153)
			6,11 (157)

Теперь можно пометить как выигрышные (за два хода) некоторые из оставшихся позиций.

Важно понимать, что и почему мы считаем выигрышными и проигрышными позициями. Проигрышной считается позиция, из которой куда бы ни пошел игрок, его соперник окажется в выигрышной позиции.

В отличие от нее, выигрышной считается позиция, из которой существует хотя бы один ход, приводящий к выигрышу. То есть (для позиций, выигрывающих за два хода) ход, который приводит соперника в проигрышную позицию.

В данном случае, все три позиции первого хода I-го игрока являются выигрышными – из каждой из них можно пойти в проигрышную позицию (выделенную жирным курсивом):

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)
		2,6 (40)
		5,5 (50)
	–3,6 (45)	***2,6 (40)***
		–3,10 (109)
		0,9 (81)
	0,5 (25)	***5,5 (50)***
		0,9 (81)
		3,8 (73)

Обозначим все эти позиции выигрышными (подчеркиванием):

Начальное положение	Первый ход I-го игрока
–3,2 (13)	2,2 (16)
	–3,6 (45)
	0,5 (25)

Получается, что начальная позиция является проигрышной (куда бы из нее ни пошел, приводишь соперника в выигрышную позицию). Значит, игрок, который ходит первым – проигрывает. Значит, выигрывает второй игрок:

Начальное положение	Первый ход I-го игрока
–3,2 (13)	2,2 (16)
	–3,6 (45)
	0,5 (25)

Мы получили ответ на первый вопрос – кто выиграет (второй игрок). Теперь нужно определить, как он должен для этого ходить. Это можно получить из этой же самой таблицы дерева игры. Напомним, на данный момент наша таблица полного дерева игры выглядит так:

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
–3,2 (13)	2,2 (16)	7,2 (53)	**12,2 (148)**	–
			7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
		2,6 (40)	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		5,5 (50)	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
	–3,6 (45)	***2,6 (40)***	7,6 (85)	**12,6 (180)**
				7,10 (149)
				10,9 (181)
			2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
		–3,10 (109)	2,10 (104)	**7,10 (149)**
				2,14 (200)
				5,13 (194)
			–3,14 (205)	–
			0,13 (169)	–

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
	0,5 (25)	***5,5 (50)***	10,5 (125)	**15,5 (250)**
				10,9 (181)
				13,8 (233)
			5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
		0,9 (81)	5,9 (106)	**10,9 (181)**
				5,13 (194)
				8,12 (208)
			0,13 (169)	–
			3,12 (153)	–
		3,8 (73)	8,8 (128)	**13,8 (233)**
				8,12 (208)
				11,11 (242)
			3,12 (153)	–
			6,11 (157)	–

Из каждой позиции, выделенной подчеркиванием нужно ходить в позицию, выделенную жирным курсивом (или просто жирным, на последнем ходу).

Удалим те позиции, в которых игра заведомо не окажется:

- выигрывающий игрок не будет во вред себе ходить неправильно (в частности, это соображение привело к удалению целых веток дерева, возникающих при ошибочных первых ходах II-го игрока (таких как 7,2; –3,10; 0,9; 3,8));
- ходы проигрывающего игрока нужно привести все, ибо ему терять нечего, и он может ходить как угодно (пытаясь, вероятно, запутать соперника ☺);
- в тех случаях, когда выигрывающий игрок имеет несколько вариантов выигрывающего хода, оставим один из них:

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока
–3,2 (13)	2,2 (16)	***2,6 (40)***	7,6 (85)	**12,6 (180)**
			2,10 (104)	**7,10 (149)**
			5,9 (106)	**10,9 (181)**
	–3,6 (45)	***2,6 (40)***	7,6 (85)	**12,6 (180)**
			2,10 (104)	**7,10 (149)**
			5,9 (106)	**10,9 (181)**
	0,5 (25)	***5,5 (50)***	10,5 (125)	**15,5 (250)**
			5,9 (106)	**10,9 (181)**
			8,8 (128)	**13,8 (233)**

Эта таблица (назовем ее "неполное дерево игры") содержит все возможные ходы первого игрока и соответствующие им ответы второго игрока. То есть, полную стратегию выигрыша. Именно эту таблицу ожидают увидеть от нас авторы-разработчики задания, именно она приведена как образец в критериях оценивания. Правда, без указания квадратов расстояния до точки (0,0).

Достаточно привести в ответе эту таблицу (с указанием того, что выигрывает второй игрок) – и мы получим наши заслуженные … 2 балла из трех.

Как же так, почему только 2!?

Потому что мы не привели доказательство правильности нашего решения. То есть, обоснованные умозаключения, из которых можно сделать вывод о правильности решения.

Это значит, нам нужно привести такие доводы, из которых будет четко понятно, почему это решение правильное и не забыть написать вывод, который мы делаем из анализа этих доводов.

Самым коротким правильным оформлением ответа, с доказательством правильности, мы считаем именно тот, который приведен в качестве примера оформления в критериях оценивания – в нем неполное дерево игры служит одновременно и указанием стратегии игры и доказательством правильности решения.

Приведем его:

Выигрывает второй игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход
Стартовая позиция	**I-й игрок (все варианты хода)**	**II-й игрок (выигрышный ход)**	**I-й игрок (все варианты хода)**	**II-й игрок (выигрышный ход)**
–3,2	**0,5**	**5,5**	**8,8**	**13,8**
			5,9	**10,9**
			10,5	**15,5**
	–3,6	**2,6**	**5,9**	**10,9**
			2,10	**7,10**
			7,6	**12,6**
	2,2	**2,6**	**5,9**	**10,9**
			2,10	**7,10**
			7,6	**12,6**

Таблица содержит *все возможные* варианты ходов первого игрока. Из нее видно, что при любом ходе первого игрока у второго имеется ход, приводящий к победе.

Это самая короткая запись ответа и поэтому почти каждая буква в нем очень важна. Рекомендуем вам выучить наизусть приведенные здесь фразы и дословно воспроизвести их на экзамене. Важно также не забыть надписать столбцы, не забыв указать, чей

это ход и что это "все варианты хода" проигрывающего игрока и "выигрышный ход" выигрывающего.

Если вам не нравится выучивать образцовое решение – можете попытаться описать свой ответ словами.

В этом случае не забудьте:

1) четко написать, кто выигрывает;

2) для каждого возможного хода проигрывающего игрока написать, как должен ходить победитель (причем сделать это не только для первого хода, но и для всех остальных ходов тоже);

3) пояснить, что Вы рассмотрели (и привели) все возможные ходы проигрывающего игрока и для каждого случая указали ходы выигрывающего игрока, приводящие к победе. На основании этого Вы делаете вывод, что ваше решение верно и что выигрывает именно этот игрок.

Теперь рассмотрим решение задачи в случае, когда выигрывает первый игрок.

При этом воспользуемся одним из методов ускорения решения – вместо вычисления каждый раз квадрата расстояния до точки (0,0) посчитаем для каждой целочисленной возможной координаты те точки, которые лежат на границе окружности заданного радиуса. Соответственно, если получающаяся позиция превышает оба значения одной из таких точек – значит, она находится за окружностью и расстояние до нее больше заданного.

Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (2,3). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (2x,y), или в точку с координатами (x,y+3), или в точку с координатами (x,y+4). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 14 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

Найдем точки с целочисленными координатами, лежащие возле границы окружности (внутри):

Если одна координата 14, то другая – 0. То есть, (14,0) и (0,14).

Если одна координата 13, то другая – $\sqrt{14^2-13^2} \approx 5{,}2$. То есть, (13,5) и (5,13).

Если одна координата 12, то другая – $\sqrt{14^2-12^2} \approx 7{,}2$. То есть, (12,7) и (7,12).

Если одна координата 11, то другая – $\sqrt{14^2-11^2} \approx 8{,}7$. То есть, (11,8) и (8,11).

Если одна координата 10, то другая – $\sqrt{14^2-10^2} \approx 9{,}8$. То есть, (10,9) и (9,10).

Построим полное дерево игры. При этом, если одна из координат станет от 10 до 14, сравним вторую координату с найденным граничным значением. Если она его превысит – значит, эта позиция лежит за окружностью радиуса 14 и игра на этом заканчивается. Например, к таким позициям относятся точки (16,3), (2,14), (2,15).

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока	Третий ход I-го игрока
2,3	4,3	8,3	**16,3**	–	–
			8,6	Можно не рассматривать	–
			8,7		–
		4,6	8,6	**16,6**	–
				8,9	Можно не рассматривать
				8,10	
			4,9	8,9	
				4,12	
				4,13	
			4,10	8,10	Можно не рассматривать
				4,13	
				4,14	–
		4,7	8,7	**16,7**	–
				8,10	Можно не рассматривать
				8,11	
			4,10	8,10	Можно не рассматривать
				4,13	
				4,14	–
			4,11	8,11	Можно не рассматривать
				4,14	–
				4,15	–
	2,6	4,6	8,6	**16,6**	–
				8,9	Можно не рассматривать
				8,10	
			4,9	8,9	**16,9**
					8,12
					8,13
				4,12	8,12
					4,15
					4,16
				4,13	8,13
					4,16
					4,17
			4,10	8,10	Можно не рассматривать
				4,13	
				4,14	–
		2,9	*4,9*	8,9	**16,9**
					8,12
					8,13

Начальное положение	Первый ход I-го игрока	Первый ход II-го игрока	Второй ход I-го игрока	Второй ход II-го игрока	Третий ход I-го игрока
				4,12	8,12
					4,15
					4,16
				4,13	8,13
					4,16
					4,17
			2,12	4,12	Можно не рассматривать
				2,15	–
				2,16	–
			2,13	4,13	Можно не рассматривать
				2,16	–
				2,17	–
		2,10	4,10	Можно не рассматривать	
			2,13		
			2,14	–	
	2,7	*4,7*	8,7	**16,7**	–
				8,10	Можно не рассматривать
				8,11	
			4,10	8,10	Можно не рассматривать
				4,13	
				4,14	–
			4,11	8,11	Можно не рассматривать
				4,14	–
				4,15	–
		2,10	4,10	Можно не рассматривать	
			2,13		
			2,14	–	
		2,11	4,11	Можно не рассматривать	
			2,14	–	
			2,15	–	

При составлении дерева игры мы в этот раз использовали несколько соображений, позволяющих не строить дальше дерево вдоль некоторых веток.

Например, на втором ходу I-го игрока в некоторых случаях достигался конец игры (выигрыш). Это значит, что позиция, из которой игра пришла в это состояние, является выигрышной. Значит, можно дальше не рассматривать другие ходы из выигрышной позиции, потому что в них игра никогда не окажется – текущий игрок просто сделает выигрышный ход (это случилось в позициях (8,3), (2,10) и (2,11) первого хода II-го игрока).

На втором ходу II-го игрока такая ситуация встречается почти всегда (за исключением позиции (4,9) второго хода I-го игрока).

Это сразу позволяет сделать вывод о том, что предыдущая позиция (второго хода I-го игрока) выигрышная и не рассматривать другие ее ветви.

Мало того, в столбце второго хода I-го игрока после применения описанной тактики два раза оказалась ситуация, когда все три варианта хода помечены как выигрышные (подчеркнуты). Это позволяет сделать вывод относительно позиций, из которых эти три варианта следуют (оба раза это позиция (4,7)). То есть, эти позиции – (4,7) – проигрышные (мы обозначили их жирным курсивом). Это позволяет нам сразу сделать вывод, что позиции, которые предшествуют позициям (4,7) – выигрышные (это (2,7) и (4,3)). Поэтому мы не будем рассматривать те ветви дерева игры, которые следуют позициям (2,7) и (4,3) и остались не "оценены". Мы даже не будем думать, выигрышные они или проигрышные, потому что в процессе игры эти ветви никогда не будут "играть" (конечно, это все действует при условии безошибочной игры выигрывающего игрока).

Остается только рассмотреть ветви (4,9) на третьем ходу I-го игрока.

Для каждого из последующих ходов ((8,9), (4,12) и (4,13)) есть хотя бы один ход, оканчивающий игру ((16,9), (4,15) и (4,16), например). Значит, все эти три позиции – (8,9), (4,12) и (4,13) – выигрышные. Отсюда можно сделать вывод, что позиция (4,9) – проигрышная (все три варианта хода из нее приводят противника к выигрышу).

Отсюда можно сделать вывод, что позиции, из которых можно придти в (4,9) – выигрышные. Это позиции (2,9) и (4,6).

Теперь оказывается, что все три варианта хода для позиции (2,6) – (4,6), (2,9) и (2,10) – помечены как выигрышные. Значит, позиция (2,6) – проигрышная.

Это позволяет сделать вывод, что стартовая позиция (2,3) – выигрышная.

Значит, первый игрок при безошибочной игре выигрывает.

Для этого первым ходом он должен пойти в позицию (2,6).

Удалим из построенного дерева игры все случаи проигрышных ходов выигрывающего игрока. Если выигрышных ходов есть несколько вариантов – оставим только один.

А вот все возможные варианты хода проигрывающего игрока оставим.

Получится следующее неполное дерево игры:

<table>
<tr><th>Начальное положение</th><th>Первый ход I-го игрока</th><th>Первый ход II-го игрока</th><th>Второй ход I-го игрока</th><th>Второй ход II-го игрока</th><th>Третий ход I-го игрока</th></tr>
<tr><td rowspan="7">2,3</td><td rowspan="7">2,6</td><td rowspan="3">4,6</td><td rowspan="3">4,9</td><td>8,9</td><td>16,9</td></tr>
<tr><td>4,12</td><td>4,15</td></tr>
<tr><td>4,13</td><td>4,16</td></tr>
<tr><td rowspan="3">2,9</td><td rowspan="3">4,9</td><td>8,9</td><td>16,9</td></tr>
<tr><td>4,12</td><td>4,15</td></tr>
<tr><td>4,13</td><td>4,16</td></tr>
<tr><td>2,10</td><td>2,14</td><td colspan="2">–</td></tr>
</table>

Напомним, как рекомендуется оформить в этом случае запись ответа:

Выигрывает первый игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	**I-й игрок (выигрышный ход)**	**II-й игрок (все варианты хода)**	**I-й игрок (выигрышный ход)**	**II-й игрок (все варианты хода)**	**I-й игрок (выигрышный ход)**
2,3	**2,6**	**4,6**	**4,9**	**8,9**	**16,9**
				4,12	**4,15**
				4,13	**4,16**
		2,9	**4,9**	**8,9**	**16,9**
				4,12	**4,15**
				4,13	**4,16**
		2,10	**2,14**	–	

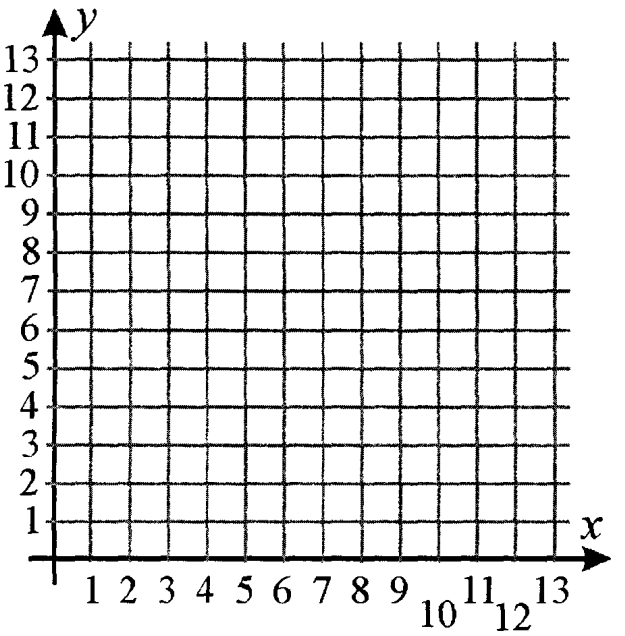
Таблица содержит *все возможные* варианты ответных ходов второго игрока на выигрышный ход первого игрока. Из нее видно, что при любом ходе второго игрока у первого имеется ход, приводящий к победе.

Для данной задачи (с фишкой на координатной плоскости) можно применить графический метод решения. Смысл его примерно тот же – сначала рассматриваются все позиции, которые в процессе игры достижимы из стартовой. Потом, начиная "с конца", от тех позиций, которые лежат возле окружности заданного радиуса и про которые можно сразу сказать, что они выигрышные, постепенно, приближаясь к стартовой позиции, оцениваются на выигрышность/проигрышность остальные позиции. В итоге делается вывод о выигрышности/проигрышности стартовой позиции и по полученной схеме строится неполное дерево игры (которое нужно для записи ответа).

Рассмотрим графическое решение на примере задачи:

Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (1,3). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+4,y), или в точку с координатами (x,y+3), или в точку с координатами (x,y+4). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 13 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

Сначала нарисуем координатную плоскость, на которой можно будет изобразить все возможные позиции игры (в данном случае, достаточно координатной плоскости в I-й четверти, размером 13 единиц по осям X и Y):



Найдем точки с целочисленными координатами, лежащие возле границы окружности.

Если одна координата 13, то другая – 0. То есть, (13,0) и (0,13).

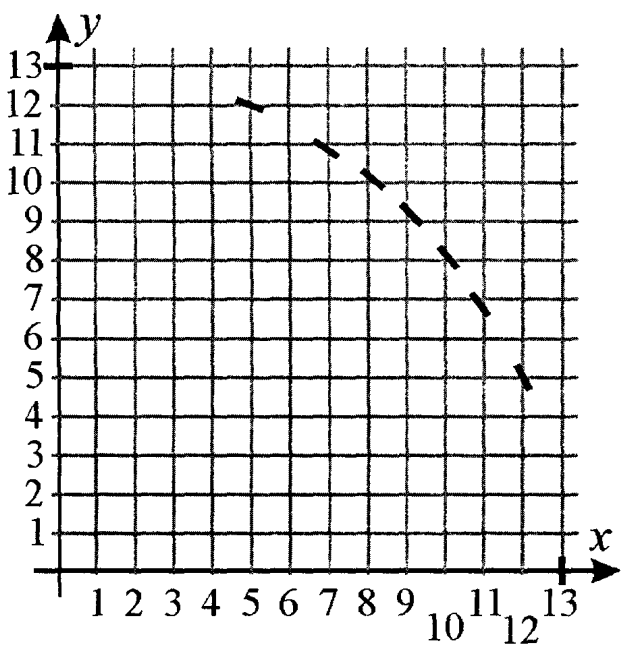
Если одна координата 12, то другая – $\sqrt{13^2-12^2}=5$. То есть, (12,5) и (5,12).

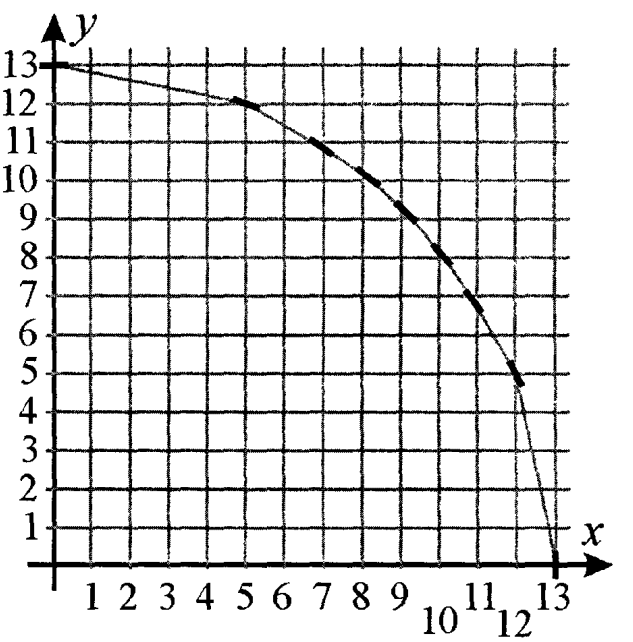
Если одна координата 11, то другая – $\sqrt{13^2-11^2}\approx 6{,}9$. То есть, (11,6) и (6,11).

Если одна координата 10, то другая – $\sqrt{13^2-10^2}\approx 8{,}3$. То есть, (10,8) и (8,10).

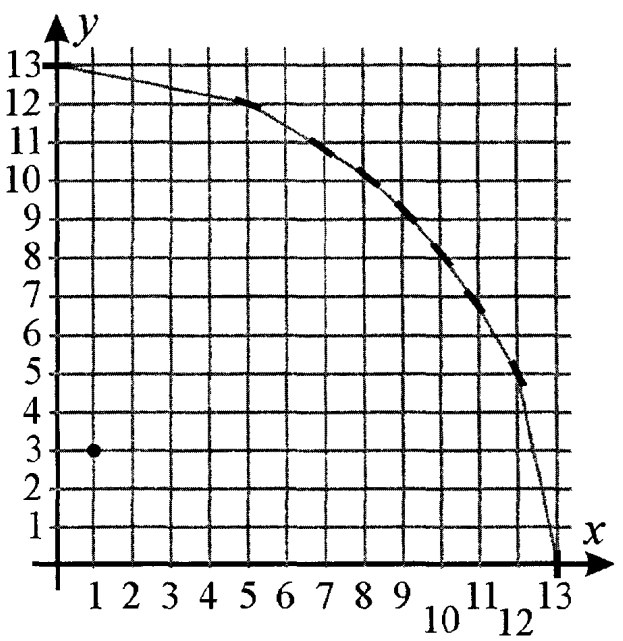
Если одна координата 9, то другая – $\sqrt{13^2-9^2}\approx 9{,}4$. То есть, (9,9).

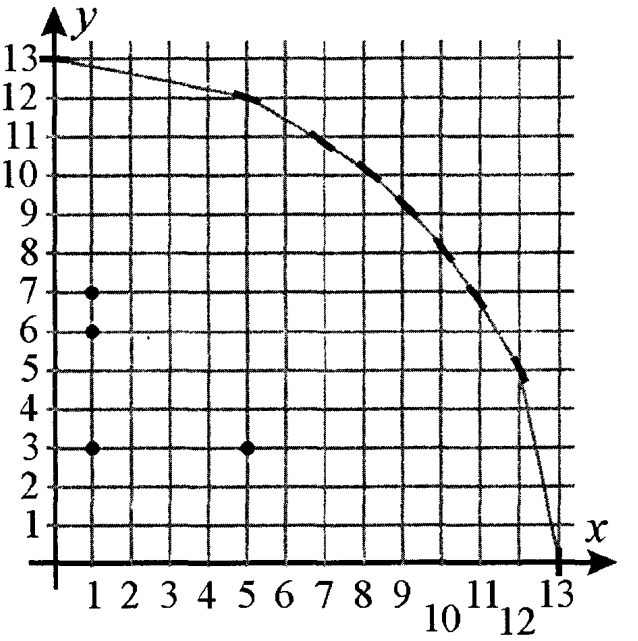
Обозначим штрихами точки, через которые пройдет окружность радиуса 13 и нарисуем ее. Для рисовании окружности просто соединим штрихи прямыми отрезками.



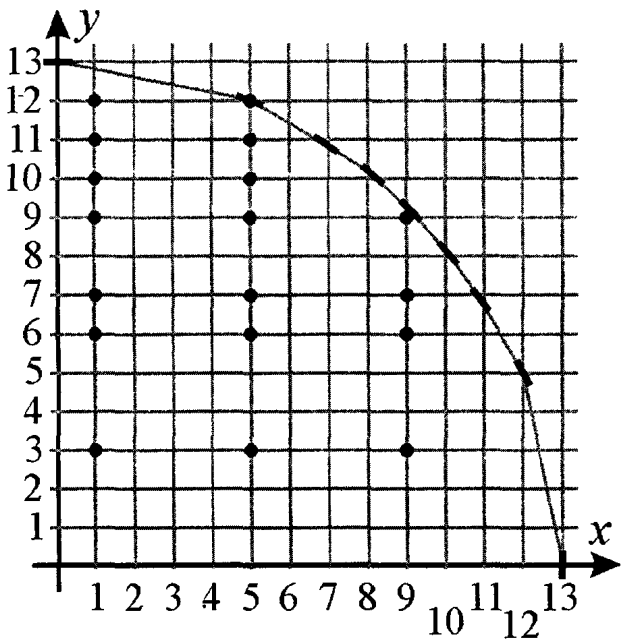


Теперь обозначим стартовую позицию и позиции, которые достижимы из стартовой:

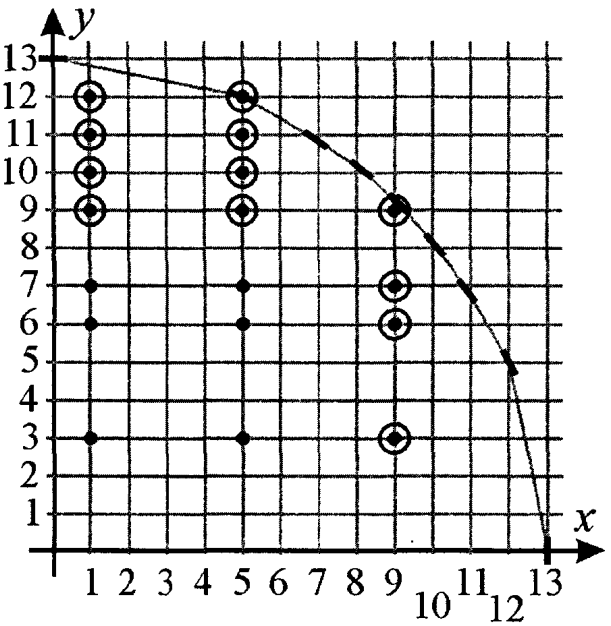




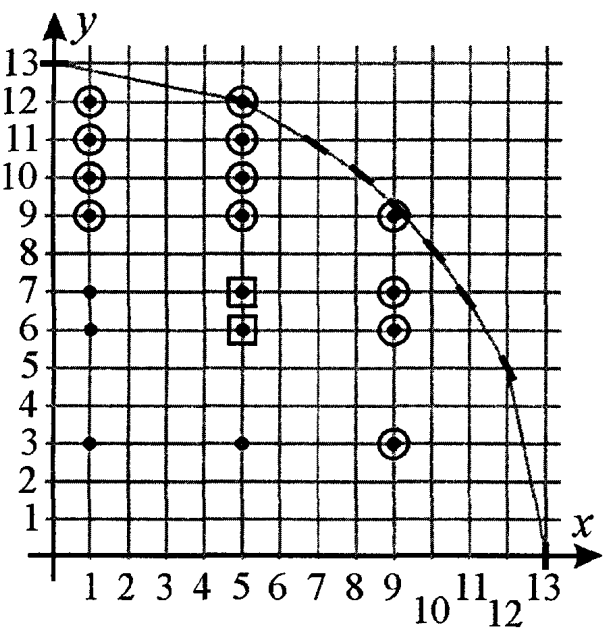
Затем, аналогично, обозначим все остальные позиции, которые достижимы в процессе игры и не выходят за нарисованную окружность:



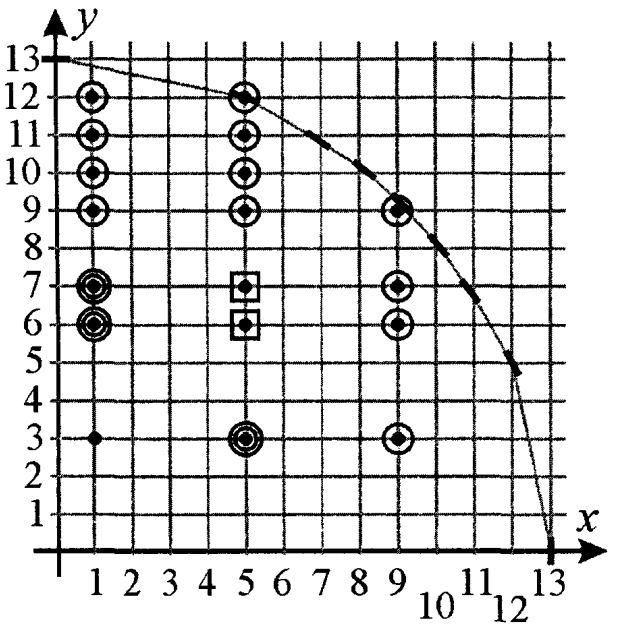
Теперь все позиции, из которых можно одним ходом "выйти" за окружность, обозначим как выигрышные (на схеме – обведем их в кружок):



Теперь найдем те позиции, из которых любой ход приводит игру в выигрышную позицию. На схеме – это те позиции, у которых справа в четырех единицах, а сверху – в трех и четырех единицах нарисован кружок. Таких позиций в данном случае нашлось две: (5,7) и (5,6). Обозначим их как проигрышные (обведем квадратиком):

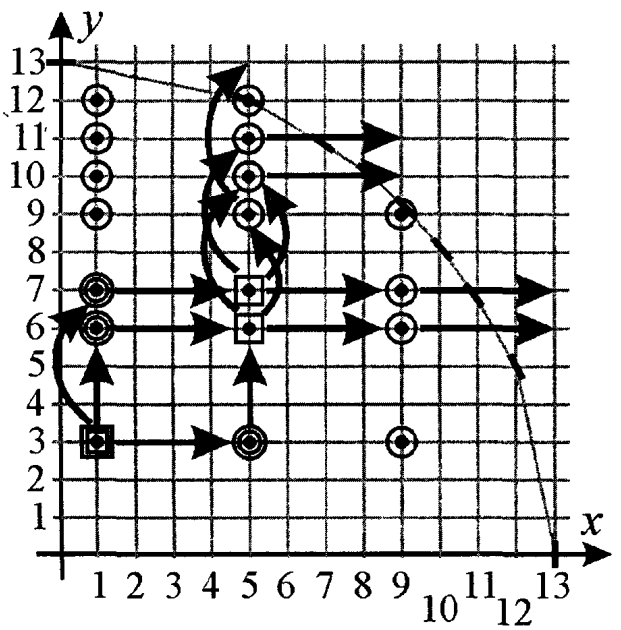


Теперь можно найти те позиции, из которых можно попасть в проигрышные (то есть, позиции, выигрышные за два хода). Это те позиции, которые находятся слева от квадратиков на 4 и снизу от квадратиков на 3 и на 4 единицы. В данном случае, это позиции (1,6), (1,7) и (5,3). Обозначим их как выигрышные (обведем кружком, для удобства – двойным кружком):



Из получившейся схемы видно, что стартовая позиция – проигрышная (все три возможных хода из нее обозначены как выигрышные). Значит, первый игрок проигрывает. То есть, выигрывает второй игрок.

По получившейся схеме можно составить схему-стратегию игры. Из проигрышных позиций будем рассматривать все три возможных хода. Из выигрышных позиций – только один выигрышный. Ходы обозначим стрелками:



Вообще говоря, полученная схема игры, со стрелками верных и возможных ходов, может считаться стратегией игры. А рассуждения, почему позиции помечены именно так, могут считаться доказательством правильности решения. Но мы рекомендуем вам не искушать судьбу (в лице эксперта, который будет проверять вашу работу ☺) и записать ответ в том виде, который приведен в демоверсии в качестве образца.

В данном случае, по схеме составляем таблицу неполного дерева игры и оформляем ответ:

Выигрывает второй игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход
Стартовая позиция	**I-й игрок (все варианты хода)**	**II-й игрок (выигрышный ход)**	**I-й игрок (все варианты хода)**	**II-й игрок (выигрышный ход)**
1,3	**5,3**	**5,6**	**9,6**	**13,6**
			5,9	**5,13**
			5,10	**9,10**
	1,6	**5,6**	**9,6**	**13,6**
			5,9	**5,13**
			5,10	**9,10**
	1,7	**5,7**	**9,7**	**13,7**
			5,10	**9,10**
			5,11	**9,11**

Таблица содержит *все возможные* варианты ходов первого игрока. Из нее видно, что при любом ходе первого игрока у второго имеется ход, приводящий к победе.

Задачи для самостоятельного решения

3.1. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (3,2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+3,y), или в точку с координатами (x,y+3), или в точку с координатами (x,y+4). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 13 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

3.2. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (1,2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+3,y), или в точку с координатами (x+5,y), или в точку с координатами (x,2y). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 14 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

3.3. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (0,2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+2,y), или в точку с координа-

тами (x+4,y), или в точку с координатами (x,y+3). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 13 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

3.4. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (2,2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+3,y), или в точку с координатами (x,y+2), или в точку с координатами (x,y+4). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) больше 13 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

3.5. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (2,0). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x,y) в одну из трех точек: или в точку с координатами (x+3,y), или в точку с координатами (x,y+4), или в точку с координатами (x+2,y+2). Выигрывает игрок, после хода которого расстояние по прямой от фишки до точки с координатами (0,0) не меньше 13 единиц. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

3.6. Два игрока играют в следующую игру. Перед ними лежат две кучки камней, в первой из которых 2, а во второй – 3 камня. У каждого игрока неограниченно много камней. Игроки ходят по очереди. Ход состоит в том, что игрок или удваивает число камней в какой-то куче, или добавляет 4 камня в какую-то кучу. Выигрывает игрок, после хода которого в одной из куч становится не менее 20 камней. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Как должен ходить выигрывающий игрок? Ответ обоснуйте.

3.7. Два игрока играют в следующую игру. Перед ними лежат две кучки камней, в первой из которых 3, а во второй – 4 камня. У каждого игрока неограниченно много камней. Игроки ходят по очереди. Ход состоит в том, что игрок или удваивает число камней в какой-то куче, или добавляет 3 камня в какую-то кучу. Выигрывает игрок, после хода которого сумма камней в обоих кучах становится не менее 20 камней. Кто выигрывает при безошибочной игре обоих игроков – игрок, делающий первый ход, или игрок, делающий второй ход? Как должен ходить выигрывающий игрок? Ответ обоснуйте.

Рекомендация по решению задач 3.6 и 3.7

При построении полного дерева игры в этих заданиях дерево получается достаточно громоздким. Для упрощения решения предлагаем воспользоваться графическим способом или при построении дерева игры заранее определить математические условия для выигрышных позиций. Это позволит при построении дерева не рисовать последний уровень. То есть, сократит рисуемое дерево в два раза. Так, например, в задаче 3.6 выигрышной позицией можно считать такую, при которой в одной куче становится не менее 10 камней.

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ. СОЗДАНИЕ ПРОГРАММ ДЛЯ РЕШЕНИЯ ЗАДАЧ СРЕДНЕЙ СЛОЖНОСТИ

Характеристика задания с развернутым ответом С4 ЕГЭ по информатике

В соответствии со спецификацией экзамена, задание С4 нацелено на проверку умения создавать собственные программы (30–50 строк) для решения задач средней сложности. Это задание отличается от других заданий ЕГЭ по информатике максимальным первичным баллом – 4 балла и наибольшим запланированным временем – один час из четырех часов экзамена. В этом задании проверяется не только умение составить алгоритм, но и написать законченную программу на одном из языков программирования (по выбору экзаменуемого), т.е. владение технологией программирования.

Внимательное изучение спецификации экзамена может привести к вопросу: "Почему задача на составление программы охарактеризована как имеющая среднюю сложность, а самому заданию С4 присвоена высшая категория сложности? Нет ли здесь противоречия?". На самом деле, противоречие здесь кажущееся. Действительно, проверяемый заданием С4 элемент подготовки – умение решать программистские задачи среднего уровня, но само это задание относится к высокому для выпускника уровню сложности, что подтверждается статистикой его выполнения. Так, в 2008 году высокие баллы (3 или 4) за задание С4 получили всего 5,4% участников экзамена. В первой волне ЕГЭ 2009 г.[4] результаты выполнения задания С4 были следующими:

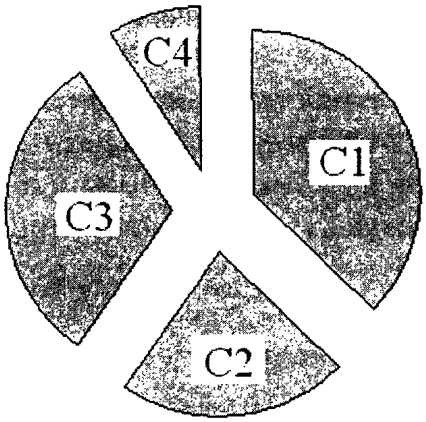
Балл	0	1	2	3	4
% участников, набравших данный балл	85,7	5,1	4,9	3,0	1,3

Т.е. ненулевой результат показали только 14,3% участников.

Это вполне нормальная статистика для задания высокого уровня сложности, которое вовсе не предназначено для массового успешного выполнения. Напротив, цель сложного задания – дать возможность наиболее подготовленным учащимся повысить свой итоговый балл на задаче, соответствующей вузовским требованиям к абитуриенту с углубленной подготовкой по информатике.

Для того, чтобы проиллюстрировать относительную сложность заданий части С, приведем на круговой диаграмме соотношение участников экзамена, получивших ненулевые баллы за задания части С:

[4] Здесь и далее статистические данные по Москве (более 4800 участников)



В виде таблицы эти данные выглядят следующим образом:

Задание	C1	C2	C3	C4
% участников, набравших ненулевой балл за задание	57,7	34,0	48,6	14,3

Сумма чисел в строке, естественно, больше 100, поскольку многие участники получали ненулевые баллы сразу за несколько заданий части С.

Можно сделать вывод о том, что задание С4 действительно является самым сложным заданием экзамена.

Характеристика «задача средней сложности на программирование» относится, безусловно, к учебному школьному программированию и означает, что для ее решения нет необходимости привлекать ни методы динамического программирования, ни структуры данных, по сложности превосходящие массивы, хотя умение работать с такими структурами данных как стеки, линейные списки, деревья подразумевается в профильном стандарте по информатике. В тоже время, для успешного выполнения задания С4 требуются не только навыки алгоритмизации, но и свободное владение подмножеством языка программирования, включающим описание типов данных, работу с массивами, операции с числовыми, логическими, символьными и строковыми данными, ввод-вывод данных, организацию ветвлений и циклов.

Уровень сложности задания С4 представляется разумным компромиссом между требованиями ведущих ВУЗов, желающих максимально дифференцировать «хороших», «очень хороших» и «отличных» абитуриентов по способности к самостоятельному практическому программированию, и возможностями школ, зачастую ограниченными, по обучению старшеклассников программированию на языке высокого уровня.

Типичное условие задания С4, его анализ

Типичная постановка задачи С4, как и любой задачи по программированию, содержит:

- формат входных данных;
- назначение программы, т.е, какую итоговую информацию программа должна извлечь из исходных данных или как их преобразовать;
- формат выходных данных;
- дополнительные условия и рекомендации программисту.

Приведем пример типичного условия задачи С4 из проекта демо-версии 2010 года:

На автозаправочных станциях (АЗС) продается бензин с маркировкой 92, 95 и 98. В городе М был проведен мониторинг цены бензина на различных АЗС.

Напишите эффективную по времени работы и по используемой памяти программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет определять для каждого вида бензина, сколько АЗС продают его дешевле всего. с N - число строк данных о стоимости бензина. В каждой из следующих N строк находится информация в следующем формате:

<Компания> <Улица> <Марка> <Цена>,

где <Компания> – строка, состоящая не более, чем из 20 символов без пробелов,
<Улица> – строка, состоящая не более, чем из 20 символов без пробелов,
<Марка> – одно из чисел – 92, 95 или 98,
<Цена> – целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках.

<Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <цена> разделены ровно одним пробелом. Пример входной строки:

МигОйл Мичуринский 95 2250

Программа должна выводить через пробел 3 числа – количество АЗС, продающих дешевле всего 92-й, 95-й и 98-й бензин соответственно. Если бензин какой-то марки нигде не продавался, то следует вывести 0. Пример выходных данных:

12 1 0

Итак, выделим основные элементы условия.

Формат входных данных:

На вход программе сначала подается N – число строк данных о стоимости бензина. В каждой из следующих N строк находится информация в следующем формате:

<Компания> <Улица> <Марка> <Цена>,

где <Компания> – строка, состоящая не более, чем из 20 символов без пробелов,
<Улица> – строка, состоящая не более, чем из 20 символов без пробелов,
<Марка> – одно из чисел – 92, 95 или 98,
<Цена> – целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках.

<Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <цена> разделены ровно одним пробелом. Пример входной строки:

МигОйл Мичуринский 95 2250

Назначение программы:

Напишите … программу, которая будет определять для каждого вида бензина, сколько АЗС продают его дешевле всего.

Формат выходных данных:

Программа должна выводить через пробел 3 числа – количество АЗС, продающих дешевле всего 92-й, 95-й и 98-й бензин соответственно. Если бензин какой-то марки нигде не продавался, то следует вывести 0. Пример выходных данных:

12 1 0

Дополнительные условия и рекомендации программисту:

Напишите эффективную по времени работы и по используемой памяти программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0)...

Выделив основные элементы условия, мы сделали первый шаг на пути формализации задачи: от текста на естественном языке до текста программы на формальном языке – языке программирования.

Обратите внимание, что часть условия не вошла ни в один из существенных элементов. Это литературное введение, т.е. фразы, поясняющие задачу, вводящие в курс дела, но не существенные для решения задачи:

На автозаправочных станциях (АЗС) продается бензин с маркировкой 92, 95 и 98. В городе М был проведен мониторинг цены бензина на различных АЗС.

Действительно, дальше про эти два предложения можно забыть, т.к. все возможные маркировки бензина четко прописаны в формате входных данных "**...<Марка> – одно из чисел – 92, 95 или 98...**", а понятие мониторинга и идентификатор города – "M" далее нигде не фигурируют.

Такая схема разбора условия рекомендуется при дальнейшей работе с подобными задачами.

Внимательное чтение условия очень важно, т.к. значительное количество досадных ошибок учащихся связано с невнимательным чтением, неверным пониманием того или иного компонента условия.

Не вдаваясь пока в особенности решения данной задачи, поясним общие для всех заданий С4 дополнительные условия и рекомендации программисту. Что значит "**эффективная по времени работы и по используемой памяти программа**"? Эффективная по времени работы программа не должна содержать лишних операций, число которых пропорционально N – объему входных данных, или некоторой функции от N, например N^2 или $\log_2 N$. Если число лишних операций пропорционально 2^N или $N!$ – это совсем нехорошо. Например, при решении рассматриваемой задачи лишней была бы сортировка массива цен.

Что значит "**эффективная по используемой памяти программа**"? Эффективная по памяти программа не должна содержать лишних выделений памяти, размер которых пропорционален N – объему входных данных, или некоторой функции от N, например N^2 или $\log_2 N$. При этом неважно, выделяется ли эта лишняя память динамически (в том числе в результате реализации рекурсивной подпрограммы) или статически (массивом фиксированной длины).

Приведем пример идеи правильного, но неэффективного по использованию памяти решения рассматриваемой задачи:

1. при считывании исходных данных запомним цены на бензин в отдельном массиве для каждой марки бензина;
2. в цикле найдем минимум для каждого из трех массивов;
3. в следующем цикле найдем, сколько элементов в каждом массиве равны минимальному.

Это решение является неэффективным по использованию памяти, т.к. вводимые данные хранятся в дополнительном массиве. Поясним это на примере упрощенной задачи. Пусть на вход программе подается непустая последовательность из N целых чисел в диапазоне от 1000 до 3000. Напишем фрагмент программы[5], в котором напечатаем ее минимальный элемент и число повторений этого элемента в последовательности.

Неэффективное решение:

Паскаль	Бейсик
Var A: Array [1..10000] Of Integer; I,N, Min, Count: Integer; Begin ReadLn (N); for I:=1 to N do ReadLn (A[I]); Min := 3001;*{все значения по условию заведомо меньше Min}* For I:=1 To N do If (A[I] < Min) then Min := A[i]; Count := 0; For I:=1 To N Do If (A[I] = Min) Then Count := Count+1; WriteLn (Min, Count); End.	DIM A(10000) AS INTEGER INPUT N FOR I=1 To N INPUT A(I) NEXT I Min = 3001 *REM Все значения по условию* *REM заведомо меньше Min* FOR I=1 TO N IF (A(I) < Min) THEN Min = A(i) INDIF NEXT I Count = 0 FOR I=1 TO N IF (A(I) = Min) THEN Count = Count+1 ENDIF NEXT I PRINT Min, Count

Эффективное решение:

Паскаль	Бейсик
Var A, I, N, Min, Count: Integer; Begin Count := 0; Min := 3001; ReadLn (N); For I:=1 To N Do Begin ReadLn (A); If (A = Min) Then Count := Count+1; If (A < Min) Then Begin Min := A; Count := 1; End; End; WriteLn (Min, Count); End.	INPUT N COUNT = 0 Min = 3001 FOR I=1 TO N INPUT A IF (A = Min) THEN Count = Count+1 ENDIF IF (A < Min) THEN Min = A Count = 1 ENDIF NEXT I PRINT Min, Count

[5] Здесь и далее при чтении текстов программ следует учитывать, что в Паскале и Бейсике строчные и прописные буквы в ключевых словах и идентификаторах не различаются, в Си – различаются.

Кстати, косвенное указание на то, что надо искать алгоритм решения, не использующий копирование входных данных в массив, можно получить, анализируя условие задачи. В условии нет ограничения на количество АЗС в городе, поэтому формально оно может быть очень большим, и статическое выделение памяти определенного размера (в примере – массив из 10000 целых чисел) тем самым исключено.

Необходимо пояснить, что требование эффективности по памяти и по времени не является неким искусственным ограничением, дополнительным "барьером" для учащегося, но отражает профессиональную практику программирования. Конечно, объемы памяти и быстродействие современных компьютеров растут очень быстро, но не стоит забывать, что объемы обрабатываемых данных в условиях всеобщей информатизации растут не медленнее, а иногда и опережающими темпами.

Вернемся к условию задачи. В нем содержится рекомендация **"укажите используемую версию языка программирования, например, Borland Pascal 7.0"**. Для чего это нужно делать? Дело в том, что образовательные стандарты не конкретизируют обязательный для изучения язык программирования. Поэтому выбор конкретного языка (языков) программирования, на примере которого раскрывается содержание тем "Алгоритмизация и программирование" и "Технология программирования» остается за учебными заведениями. Практика показывает, что в российских школах наиболее часто изучаемыми являются Pascal и Basic. Именно поэтому здесь и далее примеры программ будут приводиться на этих языках. На третьем месте по частоте изучения – C или C++. Гораздо реже встречаются такие "экзотические" для средней школы языки как C#, Java, PHP, Perl, Python, Лого (последний "экзотический" для решения задач рассматриваемого типа, хотя он и достаточно распространен в основной школе).

На экзамене учащийся может решать задание С4 на любом языке программирования, которым он владеет, его работа будет проверена и объективно оценена. Выбор конкретного языка на оценку не влияет. В системе оценивания не предусмотрены бонусы за выбор "правильного" или "оригинального" языка программирования, поэтому выпускникам, владеющим несколькими языками программирования, следует руководствоваться следующими критериями:

- каким языком лучше владеет экзаменуемый;
- какой лучше подходит для решения поставленной задачи.

Ни в коем случае не следует руководствоваться принципом "удивить экзаменационную комиссию редким или сверхновым языком программирования". Проверяющих интересует правильность предложенного решения, а не распространенность и модность языка, на котором оно выполнено.

Следует помнить, что многие языки программирования фактически представляют собой семейство из различных версий и диалектов. Естественно, в разных школах могут изучаться разные версии языков. Особенно это актуально для языков семейства Basic (QBasic, TurboBasic, FreeBasic, Visual Basic, VBA, Small Basic, Power Basic и т.д.). Паскаль тоже имеет различные версии: классический авторский Паскаль Вирта, Borland Pascal, Object Pascal (Delphi), Free Pascal, GNU Pascal, Extended Pascal и др.

Версии одного и того же языка могут отличаться как синтаксисом, так и семантикой (смыслом) некоторых конструкций языка. То, что в одном диалекте языка является ошибкой, в другом может являться нормой использования.

Так, например, в языке и системе программирования Borland Pascal предполагается, что при запуске программы из среды программирования по умолчанию все числовые пе-

ременные в момент начала выполнения программы инициализируются нулевыми значениями, поэтому фрагмент программы на Borland Pascal ввода и суммирования 100 чисел

```
var S, i, M : integer;
begin
 for i:=1 to 100 do
 begin
   readln (M);
   S:=S+M
 end;
 writeln (S)
end.
```

будет работать правильно, поскольку первоначальное значение переменной S будет по умолчанию проинициализировано нулем.

В классическом Паскале Вирта никакой инициализации по умолчанию не предусмотрено, поэтому значение переменной S перед циклом будет неопределенным, "мусорным" и результат программы будет непредсказуем. Чтобы избежать ошибки, в классическом Паскале необходимо явно проинициализировать переменную S:

```
var S, i, M : integer;
begin
 S:=0;
 for i:=1 to 100 do
 begin
   readln (M);
   S:=S+M
 end;
 writeln (S)
end.
```

Именно для того, чтобы эксперты могли правильно оценить решение в такого рода ситуациях и требуется **"указать используемую версию языка программирования, например, Borland Pascal 7.0".** Пожалуйста, не забывайте этого делать.

Замечание:

Несмотря на то, что некоторые версии языков программирования предусматривают автоматическую инициализацию переменных нулевыми значениями, этой возможностью пользоваться **крайне не рекомендуется** ни в учебном, ни в практическом программировании, поскольку очень велико число ошибок, порождаемых этим якобы "упрощением". Так, например, программа может "вдруг" перестать работать после перехода на другую версию языка, кроме того, инициализация обычно не предусмотрена для переменных, описанных в подпрограммах.

Типичные критерии оценивания задания С4

Очевидно, что целью любого участника ЕГЭ является получение максимального количества баллов. Поскольку баллы выставляются проверяющими экспертами в соответствии с критериями оценивания, необходимо ознакомиться с типичными критериями оценивания задания (во время экзамена, критерии оценивания участнику, конечно, недоступны). Сделаем это на примере все той же задачи про бензоколонки из проекта демоверсии.

Критерии оценивания начинаются с раздела «**Содержание верного ответа**»:

Программа читает все входные данные один раз, не запоминая их в массиве, размер которого соответствует числу АЗС или диапазону цен. Во время чтения данных определяются минимальная цена каждой марки бензина и количество АЗС, продающих его по этой цене. Для этого используются 6 переменных или соответствующие массивы (например, для удобства из 8 элементов каждый, см. программу на языке Бейсик).

Баллы начисляются только за программу, которая решает задачу хотя бы для одного частного случая (например, когда для каждой марки бензина минимальная цена отмечена ровно на одной АЗС).

Ниже приведены примеры решения задания на языках Бейсик и Паскаль. Допускаются решения, записанные на других языках программирования. При оценивании решений на других языках программирования необходимо учитывать особенности этих языков программирования.

Далее следуют примеры правильных решений на нескольких языках программирования:

Пример правильной и эффективной программы на языке Паскаль:

```
var
  min, ans: array[92..98] of integer;
  c: char;
  i, k, N, b: integer;
begin
  for i:=92 to 98 do
    begin
      min[i]:=3001;{допустимо и другое число, >3000}
      ans[i]:=0;
    end;
  readln(N);
  for i:=1 to N do
  begin
    repeat
      read(c);
    until c=' '; {считана компания}
    repeat
      read(c);
    until c=' '; {считана улица}
    readln(k,b);
    if min[k] > b then
      begin
        min[k]:=b;
        ans[k]:=1
      end else
    if min[k] = b then
      ans[k]:=ans[k]+1;
  end;
  {если бензина какой-то марки не было,
   ans[i] осталось равным 0}
  writeln(ans[92],' ', ans[95],' ', ans[98])
end.
```

Пример правильной и эффективной программы на языке Бейсик:

```
DIM min(8) AS INTEGER, ans(8) AS INTEGER
DIM s AS STRING
FOR i = 2 TO 8
min(i) = 3001
ans(i) = 0
NEXT i
INPUT n
FOR j = 1 TO n
LINE INPUT s
c$ = MID$(s$, 1, 1)
i = 1
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s$, i, 1)
WEND
i = i + 1
c$ = MID$(s$, i, 1)
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s$, i, 1)
WEND
i = i + 2
REM Выделим из марки бензина только последнюю цифру
k = ASC(MID$(s$, i, 1)) - ASC("0")
i = i + 2
b = VAL(MID$(s$, i))
IF min(k) > b THEN
  min(k) = b
  ans(k) = 1
 ELSE IF min(k) = b THEN ans(k) = ans(k) - 1
ENDIF
NEXT j
PRINT ans(2),ans(5),ans(8)
END
```

Следует подчеркнуть, что данные тексты программ являются лишь примерами одного из возможных решений задачи. Естественно, при оценке выполнения работы, эксперты оценивают его правильность независимо от степени похожести на приведенные авторами задания примеры программ.

Далее следуют собственно критерии оценивания, т.е. указания, за что ставить тот или иной балл. Напомним, что выполнение задания С4 оценивается максимум из 4-х баллов.

Указания по оцениванию	**Баллы**
Программа работает верно для любых входных данных произвольного размера и находит ответ, не сохраняя входные данные в массиве, размер которого соответствует числу N *(количество данных мониторинга) или диапазону цен (3000). Программа просматривает входные данные один раз, используя для нахождения ответа два массива из 3-х (8-и) элементов каждый (как в примерах программ) или 6 соответствующих переменных.*	4

Указания по оцениванию	Баллы
Допускается наличие в тексте программы одной синтаксической ошибки: 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 (если одна и та же ошибка встречается несколько раз, то это считается за одну ошибку).	
Программа работает верно, но входные данные или только цены запоминаются в массиве, в том числе возможно в массиве (трех массивах) с индексами от 0 до 3000, обозначающем количество АЗС, продающих бензин по соответствующей цене, или входные данные считываются несколько раз. Возможно, вместо алгоритма поиска минимума используется сортировка всех цен. Допускается наличие от одной до трех синтаксических ошибок: Возможно, в принципиально верно организованном вводе данных есть одна ошибка. Три балла также выставляется, если в эффективной программе, удовлетворяющей критериям выставления 4 баллов, есть одна ошибка, в результате которой программа работает не верно на некоторых (не типичных) наборах входных данных (например, все цены на одну из марок бензина равны 3000).	3
Программа работает в целом верно, эффективно или нет, но, в реализации алгоритма содержатся до двух ошибок (неверная инициализация переменных, в частности значения минимума, возможно программа не верно работает, если минимальное значение равно 3000, выход за границу массива, перевод символов в числа, используется знак "<" вместо "<=", "or" вместо "and" и т.п.). Возможно, некорректно организовано считывание входных данных. Возможно, не для всех марок бензина ответ находится верно. Допускается наличие от одной до пяти синтаксических ошибок, описанных выше.	2
Программа, возможно, неверно работает при некоторых входных данных. Возможно, программа не определяет или неверно определяет, что бензина какой-то марки не было. Или минимальная цена марки бензина считается верно, а количество соответствующих АЗС – нет. При использовании сортировки допущены ошибки в ее реализации. Допускается до 4 различных ошибок в реализации алгоритма, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи синтаксических ошибок, описанных выше.	1
Задание не выполнено или выполнено неверно	0
Максимальный балл	4

Итак, 4 балла ставится за правильную и эффективную программу максимум с одной синтаксической ошибкой.

3 балла ставится за правильную, но неэффективную программу или за правильную в целом и эффективную программу, но неверно работающую в одном из частных случаев и содержащую не более трех синтаксических ошибок.

2 балла ставится, если программа работает в целом верно, но не удовлетворяет критериям выставления 3 баллов и содержит не более двух логических и не более пяти синтаксических ошибок.

1 балл ставится, если программа не удовлетворяет критериям выставления 2 баллов и выше, и при этом содержит не более 4-х логических и семи синтаксических ошибок.

В остальных случаях ставится 0 баллов (логических ошибок более четырех, или синтаксических ошибок более семи, или задание не выполнено, или ход решения неверный).

Под синтаксическими понимаются ошибки в написании названий операторов и имен стандартных функций языка программирования, неверные расстановки и пропуски скобок, использование неописанных переменных (в тех языках, где описание переменных обязательно) т.е. все нарушения синтаксических правил языка программирования. При программировании за компьютером о синтаксических ошибках сообщает транслятор с языка программирования, при этом программа не компилируется, а в случае, если она выполняется пошагово интерпретатором – перестает выполняться.

Приведем пример фрагментов программ с синтаксическими ошибками. Фрагменты предназначены для поиска минимального значения элемента непустой входной последовательности из N элементов.

Паскаль	Бейсик
var A, N, Min: integer; begin readln (N) readln (Min); for I:=2 to N do begin readln (A); if A < Min then Min = A; end; writel (Min); end.	INPUT N INPUT Min FOR I=1 N INPUT A IF A < Min THEN Min = A ENDIF NEXT K PRINT Min

В фрагменте на Паскале имеются следующие синтаксические ошибки:

1. не описана переменная I;
2. после readln (N) отсутствует точка с запятой;
3. '=' вместо ':=' в операторе Min = A;
4. неправильно написано имя стандартной процедуры вывода – 'writel' вместо правильного 'writeln' или 'write'.

В фрагменте на Бейсике имеются следующие синтаксические ошибки:

1. 'FOR I=1 N' вместо 'FOR I=1 TO N';
2. 'NEXT K' вместо 'NEXT I';

Из критериев следует, что повторенные одинаковые синтаксические ошибки считаются за одну, то есть, если бы в программе на Паскале отсутствовали бы все символы ';', то это считалось бы за одну синтаксическую ошибку.

Логическими ошибками, как следует из их названия, являются ошибки в алгоритме, в логике программы. В отличие от синтаксических ошибок, логические ошибки не выявляются транслятором. Программа с логическими ошибками будет запускаться, не исключено, что будет внешне благополучно работать, выдавать результаты, возможно, иногда правильные. Но при некоторых исходных данных программа с логической ошибкой будет работать неверно.

Пример тех же фрагментов программ поиска минимума непустой последовательности, но уже с логической ошибкой:

Паскаль	Бейсик
var A, i, N, Min: integer; begin readln (N); Min := 0; for i:=1 to N do begin readln (A); if (A < Min) then Min := A; end; writeln (Min); end.	INPUT N Min = 0 FOR I=1 TO N INPUT A IF (A < Min) THEN Min = A ENDIF NEXT I PRINT Min

В данном случае логическая ошибка для фрагмента на Паскале и на Бейсике одна и та же – минимальное значение инициализируется нулем. Эти программы будут правильно работать только тогда, когда в последовательности есть хотя бы одно неположительное число ($A_i \leq 0$). В противном случае, если все числа последовательности положительные, то значением минимума будет ноль, т.е. число, которого вовсе нет в последовательности!

Приведем пример тех же фрагментов программ поиска минимума непустой последовательности, избавленных от ошибок:

Паскаль	Бейсик
var A, i, N, Min: integer; begin readln (N); for I:=1 to N do begin readln (A); if (i=1) or (A < Min) THEN Min := A; end; writeln (Min); end.	INPUT N FOR I=1 TO N INPUT A IF (I=1) OR (A < Min) THEN Min = A ENDIF NEXT I PRINT Min

Логические ошибки опаснее синтаксических, поскольку они не поддаются автоматическому обнаружению на этапе трансляции, приводят к неверной работе программного обеспечения, и именно поэтому в критериях оценивания С4 баллы за логические ошибки снижают сильнее, чем за синтаксические.

Технология выполнения задания С4

За несколько десятилетий существования индустрии программирования сформировались определенные правила разработки программного обеспечения. Понятно, что по сравнению с софтверными гигантами из десятков (а то и сотен) тысяч строк кода наша программа очень мала, но, тем не менее, есть общие, проверенные практикой, подходы, которые распространяются и на нее. Создание программы для решения задания С4 можно разбить на несколько этапов:

- анализ условия;
- выбор алгоритма решения задачи и языка программирования для его реализации;
- написание чернового варианта программы, его отладка и тестирование.

Пример анализа условия подробно рассмотрен в предыдущем разделе. Отметим необходимость соблюдения спецификаций ввода-вывода. Ввод и вывод данных должен осуществляться строго в предписанном формате. Его несоблюдение может быть расценено при проверке как неполное или содержащие логические ошибки решение задачи, с соответствующим понижением баллов. На этом этапе необходимо уяснить постановку задачи, отбросить лишние ("литературно-сюжетные") подробности и представить себе задачу в формальном виде. Здесь нужно быть особенно внимательным, поскольку неверная трактовка условия станет причиной создания принципиально неверной программы, которая может быть оценена даже в 0 баллов.

Приступать к написанию программы можно лишь тогда, когда детально продуман алгоритм решения задачи, не следует начинать писать программу, не представляя алгоритма ее решения. Для простой задачи решение можно держать и в голове, но лучше его зафиксировать на бумаге в виде пояснения или схемы. Так будет легче не запутаться самому, а эксперту будет проще и приятнее проверять работу.

Для написания программы нужно пользоваться тем языком программирования, которым хорошо владеешь, и который подходит для решения поставленной задачи. Не следует специально стремиться поразить экзаменаторов знанием экзотического или недавно появившегося языка программирования, поскольку дополнительные баллы за это не предусмотрены. Хорошая программа должна быть максимально простой и эффективной. Не нужно специально использовать сложные приемы или редко применяемые конструкции языка программирования, если в этом нет необходимости.

При написании текста программы следует контролировать соответствие типов переменных и применяемых к ним операций, не допускать использования неопределенных (неинициализированных) значений переменных.

Текст программы нужно писать аккуратно и разборчиво даже на черновике, структурировано, выделяя отступами уровень вложенности операторов, не скупясь на комментарии и пояснения. Отсутствие комментариев и пояснений вовсе не является признаком высокой квалификации программиста, скорее наоборот. Если текст программы получается, на ваш взгляд, очень сложным и запутанным, приостановите на время ее написание и подумайте, в чем причина этой сложности, нельзя ли найти более простое и изящное решение задачи.

После того как программа написана, необходимо выполнить ее тестирование и отладку, то есть поиск возможных ошибок и их устранение. Специфической особенностью ЕГЭ по информатике является бумажная, бескомпьютерная процедура отладки и

тестирования. Это требует особенной внимательности, которую нужно тренировать при подготовке к экзамену. При тестировании старайтесь найти ошибку в своей программе, помните, что цель тестирования и отладки – поиск и устранение ошибок, а не демонстрация правильности программы. Гораздо лучше, если ошибку найдете и исправите вы сами, а не экзаменаторы при проверке.

При бескомпьютерном тестировании (трассировке) программы можно вести на черновике таблицу значений переменных (как это было показано при описании заданий С2), но лучше натренироваться делать это в уме для экономии времени.

При тестировании следует подбирать набор тестовых данных так, чтобы на совокупности тестов выполнились все операторы, все "ветки" программы. Особое внимание следует уделить особым случаям исходных данных, пограничным, критическим точкам. В разобранном выше задании С4 из проекта демо-версии, критическими можно считать следующие случаи:

- бензин одной или нескольких марок отсутствует на всех АЗС;
- стоимость бензина достигает минимально возможных или максимально возможных значений на всех или нескольких АЗС;
- число АЗС равно нулю или единице.

Если в задании приведен пример входных данных, как в нашем случае:
МигОйл Мичуринский 95 2250,
то его тоже надо включить в состав тестов, но, очевидно, он не должен быть единственным тестом.

При трассировке необходимо еще раз внимательно проконтролировать синтаксис программы (должен быть явно указан тип каждой переменной, если этого требует язык программирования, переменные должны использоваться в соответствии со своим типом, не должно быть неинициализированных и неиспользуемых переменных, должны быть правильно расставлены операторные скобки и разделители операторов и т.д.)

В случае внесения в программу в ходе тестирования исправлений, ее следует протестировать заново.

После завершения трассировки программу нужно аккуратно переписать на чистовик. При этом желательно снабдить ее комментариями (или сохранить их, если они уже были на черновике), поясняющими работу ее отдельных блоков, нюансы работы программы.

Конечно же, не следует писать "комментарии ради самих комментариев" вроде
i:=1 ; *{присваиваем переменной i значение 1 }*
Но комментарии в стиле

```
...
Repeat
  Read(c);
Until c=' '; {считана компания}
Repeat
  Read(c);
Until c=' '; {считана улица}
...
```

вполне разумны и уместны.

Крайне желательно привести в виде схемы или словесного описания алгоритм решения задачи. Также нужно не забыть указать использованную вами версию языка программирования. Как известно, при проверке ЕГЭ черновики не проверяются и не оцениваются, поэтому нужно быть внимательным при переписывании с черновика. Если что-то будет не полностью или неверно переписано, то повысить оценку даже при правильном черновике будет невозможно по регламенту проверки и апелляции.

Приемы программирования, типичные для заданий С4

Заметим, что данное пособие не является учебником по языкам программирования, поэтому в нем обсуждаются только те свойства отдельных операторов и синтаксических конструкций языков, которые, по мнению авторов, существенны для выполнения задания С4. При этом предполагается, что читатель достаточно свободно владеет хотя бы одним языком программирования.

Ввод данных

В заданиях С4 можно выделить типичные элементы решения, в которых можно использовать стандартные программистские приемы. Одним из самых важных элементов является ввод данных. В заданиях всегда указывается источник входных данных, обычная формулировка такова: **"На вход программе подаются..."**. Эта фраза означает что программа должна читать данные из стандартного входного потока, как, например, делают консольные приложения, т.е. приложения обладающие строчным (неграфическим) интерфейсом пользователя. В Паскале для организации ввода из входного потока используются стандартные процедуры Read и ReadLn, отличающиеся тем, что ReadLn после чтения своих аргументов переходит на следующую строку и отбрасывает все значения, которые остались непрочитанными в текущей строке (если такие были). В Бейсике[6] для ввода данных применяется оператор Input – для ввода одного или нескольких значения любого типа, а также оператор Line Input – для ввода строки символов. В языке Си можно использовать функцию gets.

Рассмотрим организацию ввода данных на примере упрощенного задания С4.

Пример

На вход программе подается текст заклинания, состоящего не более чем из 200 символов, заканчивающийся точкой (символ "точка" во входных данных единственный). Оно было зашифровано Гарри Поттером следующим образом: он заменил каждую английскую букву в заклинании на следующую за ней вторую по счету в алфавите (алфавит считается циклическим, то есть за буквой *Z* следует буква *A*), оставив другие символы неизменными. Строчные буквы при этом остались строчными, а прописные – прописными. Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет выводить на экран текст расшифрованного заклинания. Например, если зашифрованный текст был таким:

Bd Tc Ec Fcd Tc.

то результат расшифровки должен быть следующим:

Zb Ra Ca Dab Ra.

В данном случае можно поступить двумя способами

1) Считывать всю строку целиком в некоторую переменную строкового и типа и далее обрабатывать ее посимвольно.

2) Читать исходные данные посимвольно и сразу их посимвольно обрабатывать.

---

[6] Здесь и далее под Бейсиком понимается подмножество широко распространенных диалектов Бейсика.

Рассмотрим сначала реализацию первого способа ввода на Турбо-Паскале.

```
var str : string;
      i:integer;
begin
 readln (str);
 i:=1;
 while str[i] <> '.' do
 begin
  {Обработка входной строки, которую мы рассмотрим позднее}
   i:=i+1;
 end
end.
```

Из примера видно, что строковый тип String Турбо-Паскаля позволяет обращаться к отдельному символу строки как к элементу массива – по его индексу (номеру), при этом номера символов в строке начинаются с единицы.

На Бейсике:

```
LINE INPUT Str$
i=1
WHILE NOT (MID$(Str$, i, 1) = ".")
REM  Обработка входной строки, которую мы рассмотрим позднее
i=i+1
WEND
```

Напомним, что в Бейсике знак доллара '$' на конце идентификатора декларирует принадлежность обозначаемого им объекта к строковому типу. Для доступа к отдельному элементу строки по его номеру используется строковая функция MID$ (от «middle» – середина, нечто промежуточное, внутреннее). Первый аргумент этой функции – сама исходная строка, второй – номер символа, третий – количество подряд идущих символов возвращаемых функцией MID$, начиная с заданного. Таким образом, функция MID$ может быть использована для выделения целой подстроки, а не отдельного символа (подстроки единичной длины) как в нашем случае. Как и в Паскале, номера символов в строке в Бейсике начинаются с единицы.

Пример на Си:

```
#include <stdio.h> /*включаем описания функций ввода-
вывода*/
int main(void) {
char Str [201] ;
int i = 0;
gets (Str); /* аргумент -- адрес массива Str*/
while (Str[i] != '.'){

/* Обработка входной строки, которую мы рассмотрим позднее */
  i++;
}
return 0;
}
```

Строки в Си представляют собой массивы символов. Одно из важных отличий от Паскаля заключается в том, что символы в строке нумеруются, начиная с нуля, а не с единицы. Кроме того, символьная строка в Си всегда должна заканчиваться символом с нулевым кодом. Именно поэтому в описании массива Str в приведенном примере, указана размерность 201, а не 200. По условию задачи число символов, включая точку, не может быть больше 200, плюс завершающий нулевой символ автоматически добавляется при чтении строки.

Рассмотрим посимвольный ввод строки на Паскале для нашей задачи:

```
var c : char;
begin
 read (c); {читаем самый первый символ}
 while c <> '.' do
 begin
   {Обработка символа,  которую мы рассмотрим позднее}
    read (c); {читаем очередной символ}
 end
end.
```

На Си:

```
#include <stdio.h>
int main(void) {
char c ;
scanf("%c",&c);/*1-й аргумент-формат ввода, 2-й-адрес переменной c*/
while (c != '.'){

  /*  Обработка символа,  которую мы рассмотрим позднее */
  scanf ("%c",  &c); {читаем очередной символ}
}
```

Обратите внимание, что в двух последних примерах оператор ввода символа встречается два раза: перед циклом и внутри цикла. Это сделано для того, чтобы избежать обработки завершающего символа (точки) внутри цикла, целиком «посвятив» его непосредственно преобразованию строки.

Пример на языке Бейсик не приводится, ввиду отсутствия в нем возможности посимвольного ввода из входной строки.

Прежде, чем мы перейдем к полному решению нашей задачи про шифр Гарри Поттера, рассмотрим организацию ввода целочисленных данных на примере другого упрощенного варианта задания С4:

Пример:

В первой строке исходных данных содержится N – количество абитуриентов, сдававших по три экзамена (русский язык, математика, физика). Далее следует N строк формата

<Оценка_Русский> <Оценка_Математика> <Оценка_Физика>

с оценками абитуриента по 100-балльной шкале (от 0 до 100 включительно). Необходимо найти и напечатать максимальный и минимальный индивидуальный суммарный балл абитуриента.

Пример входных данных:
5
93 45 15
43 77 84
95 100 87
23 0 7
97 100 69

При этих исходных данных программа должна выдать:
282 30
Один из вариантов решения на Паскале:

```
var N, i, Min, Max, s: integer;
    r, m, f : 0..100;
begin
 readln (N); {Ввод количества абитуриентов}
 Min := 100;
 Max := 0;
 for i := 1 to N do
 begin
   readln (r, m, f); {Ввод оценок}
   s:= r+m+f; {подсчет индивидуальной суммы баллов}
   if  ( Min > s) then
     Min := s;
   if  ( Max < s) then
     Max := s
 end;
 writeln (Max, Min)
end.
```

Заметим, что данную задачу можно решить и с меньшим число переменных, например, используя для ввода оценок по различным предметам одну переменную:

```
var  N, i, Min, Max, s: integer;
x : 0..100;
begin
 readln (N); {Ввод количества абитуриентов}
 Min := 100;
 Max := 0;
 for i := 1 to N do
 begin
   read (x); {Ввод 1-й оценки - по русскому}
   s:= x; {учли русский}
   read (x); {Ввод 2-й оценки - по математике}
   s:= s+x; {учли математику}
   readln (x); {Ввод 3-й оценки - по физике}
```

```
    s:= s+x;{подсчет индивидуальной суммы баллов, учли, наконец, и физику}
    if   ( Min > s) then
      Min := s;
    if   ( Max < s) then
      Max := s
  end;
  writeln (Max, Min)
end.
```

Однако, такую "экономию" нельзя назвать полезной даже не потому, что при сокращении трех переменных появилось четыре дополнительных строки и два вызова процедур ввода, а потому, что она не влияет на эффективность работы программы ни по памяти, ни по времени по рассмотренным выше критериям. Еще раз напомним, что экономия памяти заключается не в плюшкинском скаредном вылавливании "лишних" байтов, а в отсутствии неоправданного использования больших или потенциально больших массивов входных и промежуточных данных.

Один из вариантов решения на Си:

```
#include <stdio.h>
int main(void) {
int  N, i, Min, Max, s, r, m, f ;
scanf ("%d", &N); /*Ввод количества абитуриентов*/
Min = 100;
Max = 0;
for (i = 1; i<=N; i++)
{
  scanf ("%d %d %d", &r,&m,&f); /*Ввод оценок*/
  s = r+m+f;
  if   ( Min > s)
    Min = s;
  if   ( Max < s)
    Max = s;
}
printf ("%d %d \n", Max, Min);
return 0;
}
```

Обратите внимание, что в функцию `printf` передаются не адреса, а значения выводимых переменных. Символ '`\n`' в формате вывода означает переход на новую строку (аналог `writeln` в Паскале).

Как уже было ранее сказано, оператор Input Бейсика устроен так, что если мы хотим ввести несколько числовых значений, то они должны быть разделены запятыми. Но в условии задачи никаких запятых между значениями оценок нет. Самовольно изменить условие задачи мы не можем, поэтому пойдем обходным путем: будем считывать строки с отметками целиком как символьные строки, извлекать из них подстроки с оценками и преобразовывать их в целочисленные значения.

Один из вариантов решения на Бейсике:

```
INPUT N 'Ввод количества абитуриентов
Min = 100
Max = 0
FOR i = 1 TO N
LINE INPUT  Str$ 'Ввод строки с тремя оценками
REM Разбор строки
j= 0
DO
j = j + 1
c$ = MID$(Str$, j, 1)
LOOP WHILE c$ <> " "
r = VAL (Str$, 1,  j-1) ' Получили оценку по русскому
Str = MID$ (Str$, j, LEN(Str$) - j +1)
REM "Отрезаем" слева от строки прочитанные символы
j=0
DO
j = j + 1
c$ = MID$(Str$, j, 1)
LOOP WHILE c$ <> " "
m = VAL (Str$, 1,  j-1) ' Получили оценку по математике
Str$ = MID$ (Str$, j, LEN(Str$) - j +1)
REM "Отрезаем" слева от строки прочитанные символы

f = VAL (Str$, 1, Len (Str$)) ' Получили оценку по физике
s = r+m+f 'подсчет индивидуальной суммы баллов
IF  ( Min > s) THEN
Min = s
ENDIF
IF  ( Max < s) THEN
Max = s
ENDIF
NEXT i
PRINT Max, Min
```

Для выделения трех целых чисел (оценок) из текстовой строки здесь применяется один и тот же прием – ищем первый пробел после подстроки с числом в цикле с постусловием Do … Loop While, затем преобразуем подстроку с числом в собственно число с помощью функции Val, затем оставляем в строке только то, что еще не было прочитано. Напомним, что функция Len возвращает длину строки – своего аргумента.

Так, например, строка **95 100 87** будет последовательно принимать вид:

95 100 87; 100 87; 87.

Заметим, что приведенный пример программы можно было бы записать более коротко, заменив последовательность практически одинаковых кусков кода на цикл или вызов подпрограммы. Для удобства чтения примера авторы не стали этого делать.

Во многих заданиях С4 входные строки состоят из значений различных типов, например, в проекте демо-версии 2009г. данные организованы следующим образом:

В первой строке находится N - число строк данных о стоимости бензина. В каждой из следующих N строк находится информация в следующем формате:

<Компания> <Улица> <Марка> <Цена>

где <Компания> – строка, состоящая не более, чем из 20 символов без пробелов,
<Улица> – строка, состоящая не более, чем из 20 символов без пробелов,
<Марка> – одно из чисел – 92, 95 или 98,
<Цена> – целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках.

<Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <цена> разделены ровно одним пробелом. Пример входной строки:

МигОйл Мичуринский 95 2250

В конкретной задаче демо-версии данные **<Компания> <Улица>** никак не используются, но, тем не менее, их надо корректно обработать при вводе. В условии есть необходимые нам указания на длину текстовых полей и отсутствие внутри них пробелов (без такого указания задача была бы практически неразрешимой). Приведем примеры фрагментов программ, выполняющих ввод и разбор одной строки в формате

<Компания> <Улица> <Марка> <Цена>.

Проще всего выполнить разбор входной строки такого формата на языке Си.

```
...
char Street[21]; /*Не забываем добавлять один символ для признака конца строки*/
char Company[21];
int Mark, Price;
...
scanf ("%s %s %d %d", Street, Company, &Mark,  &Price);
...
```

Здесь наглядно видно удобство языка Си – возможность самостоятельного определения программистом формата ввода-вывода данных.

В Паскале и Бейсике входную строку придется разбирать "вручную", используя тот факт, что элементы входных данных разделены пробелами. Применим тот же подход, что мы продемонстрировали, решая задачу про абитуриентов на Бейсике.

Фрагмент программы на Паскале:

```
var Street, Company : string;
    c: char;
    Mark, Price : integer;
...
 Company:= '';
 read(c);
 while c<> ' ' do
 begin
   Company := Company + c; {'+' в данном случае операция конкатенации, т.е. "склеивания" символьных значений}
```

```
    read(c);
  end;     {считана компания}
  Street:= ' ';
  read(c);
  while c<> ' ' do
  begin
    Street := Street + c;
    read(c);
  end;     {считана улица}
```

{Теперь в текущей входной строке остались два целых числа. Паскаль позволяет прочитать их с помощью стандартной процедуры readln, что избавляет нас от необходимости возиться с выделением подстрок, соответствующих числам и их преобразованием.}

```
readln(Mark, Price);
...
```

Таким образом, в программе на Паскале мы применили комбинированный подход – строковые данные разобрали "вручную", числовые – ввели с помощью стандартной процедуры.

Фрагмент программы на Бейсике:

```
...
LINE INPUT Str$ 'Ввод всей строки
...
j= 1
start = j  ' Фиксируем начало текстового поля
c$ = MID$(Str$, j, 1)
WHILE  c$ <> " "
j = j + 1
c$ = MID$(Str$, j, 1)
WEND
REM  j-1 - конец текстового поля, j-позиция пробела
Company$ = MID$ (Str$, start, j-start)  'считана компания
j = j+1
start = j
WHILE  c$ <> " "
j = j + 1
c$ = MID$(Str$, j, 1)
WEND
Street$ = MID$ (Str$, start, j-start)  'считана улица
j = j+1
start = j
WHILE  c$ <> " "
j = j + 1
c$ = MID$(Str$, j, 1)
WEND
```

```
Mark = VAL (MID$ (Str$, start, j-start))
REM  За последним полем ввода ("Цена") пробела нет.
REM Там просто конец строки.
start = j+1
finish = Len (Str$)
Price = VAL (MID$ (Str$, start,  finish-start +1))
...
```

Цикл, в котором ищется пробел-разделитель выглядит несколько громоздко. Его можно убрать с помощью стандартной функции поиска номера позиции начала вхождения подстроки в строку Instr в Бейсике.

Пример:

```
...
LINE INPUT Str$ 'Ввод всей строки
...
start = 1  ' Фиксируем начало текстового поля
c$ = MID$(Str$, j, 1)
finish  = INSTR (1, Str$, " ")    '  Позиция пробела
REM  1-й (необязательный) аргумент функции Instr - позиция,
REM  начиная с которой ведется поиск в строке
REM  (по умолчанию с 1-й); 2-й аргумент - строка;
REM  3-й - искомая подстрока
finish = finish - 1 ' Конец текстового поля
Company$ = MID$ (Str$, start, finish-start+1)
REM  считана компания
start = finish +2
finish  = INSTR (start, Str$, " ")
REM  Позиция второго пробела
finish = finish - 1
Street$ = MID$ (Str$ start, finish-start+1) 'считана улица
start = finish +2
finish  = INSTR (start, Str$, " ")
REM  Позиция третьего пробела
finish = finish - 1
Mark = VAL (MID$ (Str$, start, finish-start+1))
start = finish +2
REM  За последним полем ввода ("Цена") пробела нет.
REM  Там просто конец строки.
finish = LEN (Str$)
Price = VAL (MID$ (Str$, start,  finish-start +1))
...
```

В Турбо Паскале аналогом функции Бейсика Instr является функция Pos, но она ищет подстроку всегда с начала строки.

В заданиях С4 также встречаются исходные данные, которые имеют свой внутренний формат – обычно комбинацию целочисленного и строкового типа, возможно, с использованием некоторых разделителей.

Примеры:

<Школьный класс> – номер класса из одной или двух цифр от 0 до 12 и вплотную за ним буква класса – 10Б или 1а (возможность использования строчных и прописных букв должна быть оговорена в условии)

<Время чч:мм> – два двузначных числа из соответствующего диапазона, разделенных двоеточием, например, – 17:45 или 20:07. Как правильно записывается время "6 часов 15 минут" 6:15 или 06:15 – должно быть оговорено в условии.

<Дата дд.мм.гг > – двузначные числа из соответствующего диапазона, разделенные точками, например, 01.01.99 или 11.11.00.

Паскаль и Бейсик не предоставляют стандартных средств для ввода данных в таких форматах, поэтому данные сложной структуры следует вводить в виде строки символов (или последовательности из отдельных символов) с последующей обработкой (разбором) в том случае, если программе нужно получить доступ к отдельным элементам комбинированных данных.

В языке Си, как мы уже убедились, проблем со вводом гораздо меньше.

Рассмотрим разбор поля **<Школьный класс>** с выделением номера класса как целого числа и буквы класса как символа. Будем считать, что это поле содержится в отдельной строке исходных данных.

Фрагмент программы на языке Си:

```
...
int  N;
char letter;
scanf ("%d%c", &N, &letter); /* в N попал номер,  в letter
- буква класса */
...
```

Фрагмент программы на языке Паскаль:

```
var c, letter: char;
      N: 1..12;
...
 read (c); {1-й символ}
 N := Ord(c) - Ord ('0'); {в N - число, равное значению
первой цифры}
 read (c) ; {2-й символ}
 if c in ['0'..'9'] Then  {если 2-й символ - цифра}
 begin
   N := N*10 + ord(c) - ord ('0'); {Вычисляем номер класса}
   read (c); {и читаем следующий символ - букву класса}
 end;
 Letter := c; {в Letter - буква класса}
...
```

Этот фрагмент требует пояснений.

В Турбо Паскале, как и в Бейсике есть стандартное средство преобразования символьной строки в число – процедура Val. Мы могли бы найти в строке границы числа, вырезать его в отдельную текстовую строку и вызвать эту процедуру, подобно тому, как это делалось в примерах выше на Бейсике. Но в данном случае мы выбрали простое и красивое решение. Мы самостоятельно преобразовали символы, обозначающие цифры числа в само число.

Для этого мы воспользовались стандартной функцией Паскаля Ord, аргументом которой является некоторый символ, а результатом – ASCII код этого символа. Итак, Ord(c) – это ASCII код символа ‘c’, но нам нужен не сам ASCII код символа обозначающего цифру, а число, равное значению этой цифры. Мы знаем, что в коде ASCII символы десятичных цифр расположены подряд от ‘0’ до ‘9’. Поэтому значением выражения Ord(c) – Ord (‘0’) (где c – символ десятичной цифры) всегда будет числовое значение этой цифры. При этом нам не нужно знать значения ASCII кода ни нуля, ни остальных цифр. Такой технический прием очень часто применяется при преобразовании символьных данных в числовые "вручную".

Нам заранее неизвестно, сколько цифр в номере класса – одна или две, поэтому мы проверяем 2-й считанный символ на принадлежность множеству цифр [‘0’..’9’]. Если он цифра, то очевидно, что предыдущей символ был числом десятков, а анализируемый символ является цифрой, в которой хранится число единиц в номере класса. Поэтому значением номера класса в этом случае будет N*10 + Ord(c) – Ord (‘0’), а буквой – следующий символ.

В Бейсике аналогом функции Ord является функция ASC, тоже возвращающая ASCII код своего аргумента.

Аналогичный фрагмент программы на языке Бейсик:

```
...
LINE INPUT  Str$
...
C$ = MID$ (Str$,1,1)
N = ASC(C$)-ASC("0") 'в N - число, равное значению первой цифры
C$ = MID$ (Str$,2,1)
IF (C$>="0" ) ANd ("9" <= C$) THEN  'если 2-й символ - цифра
N = N*10 + ASC(C$) - ASC("0")
C$ = MID$ (Str$,3,1)
ENDIF
Letter$ = C$
...
```

Рассмотрим разбор поля **<Время чч:мм>** на целочисленные часы и минуты. Примем, что количество часов может быть односимвольным, т.е. "6 часов 5 минут" записывается как 6:05. Будем считать, что разбираемое поле содержится в отдельной строке исходных данных.

Фрагмент программы на языке Си:

```
...
int  h,m;
char c;
scanf ("%d%c%d",&h,&c,&m);/* в h попали часы, в m-минуты */
...
```

Фрагмент программы на языке Паскаль:

```
var c: char;
    h: 0..23;
    m: 0..59;
...
 read (c); {1-й символ}
 h := ord(c) - ord ('0');
{в h - число, равное значению первой цифры часов}
 read (c) ; {2-й символ}
 if c in ['0'..'9'] then  {если 2-й символ - цифра}
 begin
   h := h*10 + ord(c) - ord ('0'); {Вычисляем час}
   read (c);
 end;
 read (c);
 m := ord(c) - ord ('0');
{в m - число, равное значению первой цифры минут}
 read (c);
 m := m*10 + ord(c) - ord ('0'); {Вычисляем минуты}
...
```

Аналогичный фрагмент программы на языке Бейсик:

```
...
LINE INPUT  Str$
...
C$ = MID$ (Str$,1,1)
h = ASC(C$) - ASC("0")
REM  в h - число, равное значению первой цифры часов
C$ = MID$ (Str$,2,1) ' 2-й символ
IF (C$>="0" ) AND ("9" <= C$) THEN 'если 2-й символ - цифра
h = h*10 + ASC(C$) - ASC("0")
start = 4' позиция первой цифры минут
ELSE
start = 3' позиция первой цифры минут
ENDIF
C$ = MID$ (Str$, start,1)' первая цифра минут
m = ASC(C$) - ASC("0") ' m - число, равное значению первой
цифры минут
C$ = MID$ (Str$,2,1)
m = m*10 + ASC(C$) - ASC("0") ' вычислили минуты
...
```

Выполнить разбор поля <**Дата дд.мм.гг**> читателю предлагается самостоятельно в качестве упражнения.

Ввод вещественных данных числового типа схож со вводом целочисленных данных. Для Си он отличается только спецификацией формата ввода. На Паскале, как известно, тип вводимых данных должен соответствовать типу переменной в операторе

ввода. Поэтому все примеры с вводом целых чисел на Паскале справедливы и для ввода вещественных чисел, если заменить тип переменных Integer на Real.

Если в строке исходных данных есть только одно вещественное число или несколько, расположенных через запятую, то на Бейсике их можно ввести оператором Input. В противном случае, следует прочитать строку целиком, выделять подстроки, соответствующие числам, и применять к ним функцию Val.

Естественно, всегда остается возможность "ручного" преобразования последовательности символов, изображающих вещественное число, в само числовое значение, подобно тому, как мы делали с классами, часами и минутами в рассмотренных выше примерах. Читателю предлагается в качестве упражнения самостоятельно написать программы посимвольного преобразования строки в вещественное число как для представления в виде десятичной дроби, так и для экспоненциального представления.

Приведем пример реального задания ЕГЭ[7], содержащего ввод и обработку данных различных типов вместе с авторским решением и критериями оценивания.

На вход программе подаются 365 строк, которые содержат информацию о среднесуточной температуре всех дней 2005 года. Формат каждой из строк следующий: сначала записана дата в виде dd.mm (на запись номера дня и номера месяца в числовом формате отводится строго два символа, день от месяца отделен точкой), затем через пробел (для Бейсика – через запятую) записано значение температуры – число со знаком плюс или минус, с точностью до 1 цифры после десятичной точки. Данная информация отсортирована по значению температуры, то есть хронологический порядок нарушен. Требуется написать эффективную программу на любом языке программирования, которая будет выводить на экран информацию о месяцах с максимальной среднемесячной температурой. Найденные максимальные значения следует выводить в отдельной строке для каждого месяца в виде: номер месяца, значение среднемесячной температуры, округленное до одной цифры после десятичной точки.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа считывает входные данные, сразу подсчитывая в массиве, хранящем 12 вещественных чисел, сумму температур в каждом из месяцев. Затем с использованием этого массива ищется максимальная среднемесячная температура. За дополнительный просмотр среднемесячных температур (их можно как запомнить в массиве, так и вычислить заново) распечатывается информация об искомых месяцах. Баллы начисляются только за программу, которая решает задачу хотя бы для частного случая (например, месяц с максимальной температурой единственен).

[7] Здесь и далее все тексты заданий ЕГЭ взяты из открытого сегмента Федерального банка тестовых заданий.

Пример правильной и эффективной программы на языке Паскаль:

```
const d:array[1..12] of integer =
(31,28,31,30,31,30,31,31,30,31,30,31);
var m:array[1..12] of real;
    max,t:real;
    i,j:integer;
    c1,c2:char;
begin
  for j:=1 to 12 do
    m[j]:=0;
  for i:=1 to 365 do
  begin
    readln(c1,c1,c1,c1,c2,t);
    j:=(ord(c1)-ord('0'))*10+ ord(c2)-ord('0');
    m[j]:=m[j]+t
  end;
  max:=m[1]/d[1];
  for j:=2 to 12 do
    if m[j]/d[j] > max then
      max:=m[j]/d[j];
  for j:=1 to 12 do
    if abs(m[j]/d[j]-max) < 0.0001
    then
      writeln(j,' ',m[j]/d[j]:0:1)
end.
```

Пример правильной программы на языке Бейсик:

```
DATA 31,29,31,30,31,30,31,31,30,31,30,31
DIM i, j, d(12) AS INTEGER
DIM m(12)
FOR i = 1 TO 12
m(i) = 0
READ d(i)
NEXT i
FOR i = 1 TO 365
INPUT dat$, t
j=(ASC(MID$(dat$,4,1))-ASC("0"))*10+ASC(MID$(dat$,5,1))-ASC("0")
m(j) = m(j) + t
NEXT i
max = m(1) / d(1)
FOR j = 2 TO 12
IF m(j) / d(j) > max THEN
max = m(j) / d(j)
ENDIF
NEXT j
FOR j = 1 TO 12
IF ABS(m(j) / d(j) - max) < .0001 THEN
PRINT j; " ";
PRINT USING "##.#"; m(j) / d(j)
ENDIF
NEXT j
END
```

Указания по оцениванию	Баллы
Программа работает верно, т.е. определяет все месяцы, в которых среднемесячная температура максимальна, не содержит вложенных циклов (от 1 до 365 и от 1 до 12), в тексте программы не анализируется каждый месяц в отдельности (if m=1 then, if m=2 then…). При вычислении среднемесячных температур допустимо использование оператора CASE или 3-х операторов IF, для учета количества дней в том или ином месяце. Допускается наличие в тексте программы одной пунктуационной ошибки.	4
Программа работает верно, но содержит вложенные циклы (от 1 до 12 и от 1 до 365) или обрабатывает каждый месяц явным образом при считывании данных (12 операторов IF или оператор CASE, содержащий 12 вариантов). Возможно, сохраняет все входные данные в массиве. Допускается наличие от одной до трех синтаксических ошибок: пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования, не описана или неверно описана переменная, применяется операция, недопустимая для соответствующего типа данных.	3
Программа работает в целом верно, но выводит значение только одного месяца с максимальной температурой. Возможно, в реализации алгоритма содержатся 1–2 ошибки (используется знак «<« вместо «>«, «or» вместо «and» и т.п.). Допускается наличие до пяти различных синтаксических ошибок.	2
Программа неверно работает при некоторых входных данных и, возможно, содержит ошибку в алгоритме поиска максимума или среднемесячной температуры или в выделении номера месяца и температуры дня из строки входных данных. Допускается наличие до семи синтаксических ошибок.	1
Задание выполнено неверно	0
Максимальный балл	4

Обработка символьной информации

Значительное число заданий С4 связано с обработкой символьной информации. Рассмотрим некоторые стандартные приемы преобразования символьных данных. Для начала вернемся к задаче про шифр Гарри Поттера, на примере которой мы рассматривали ввод символьных данных в предыдущем разделе.

Напомним условие задачи.

На вход программе подается текст заклинания, состоящего не более чем из 200 символов, заканчивающийся точкой (символ "точка" во входных данных единственный). Оно было зашифровано Гарри Поттером следующим образом: он заменил каждую английскую букву в заклинании на следующую за ней *вторую по счету в алфавите* (алфавит считается циклическим, то есть за буквой *Z* следует буква *A*), оставив другие символы неизменными. Строчные буквы при этом остались строчными, а прописные – прописными. Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет выводить на экран текст расшифрованного заклинания. Например, если зашифрованный текст был таким:

Bd Tc Ec Fcd Tc.

то результат расшифровки должен быть следующим:

Zb Ra Ca Dab Ra.

Решение.

Организацию ввода данных мы подробно рассмотрели выше. Основной проблемой теперь является циклическая замена букв в строке. Решение с отдельной обработкой каждой буквы в стиле:

```
If  Str[i] = 'A' Then Str[i] := 'Y';
If  Str[i] = 'B' Then Str[i] := 'Z';
If  Str[i] = 'C' Then Str[i] := 'A';
...
и т.д.

или
...
Case Str[i] Of
 'A' : Str[i] := 'Y';
 'B' : Str[i] := 'Z';
 'C' : Str[i] := 'A';
 ...
 'Z' : Str[i] := 'X';
 'a' : Str[i] := 'y';
 ...

 'z' : Str[i] := 'x';
end;
 ...
```

является слишком громоздким. В практическом программировании решения такого рода плохи еще и потому, что небольшое изменение правила кодирования неизбежно приведет к изменению большого куска программы. Попробуем найти более общее решение.

Сначала рассмотрим частный случай – когда обрабатываемые буквы находятся достаточно далеко от начала алфавита и перехода через букву А (а) при расшифровке не происходит. В этом случае перекодировка очевидно сводится к замене символа с номером (кодом) N на символ с номером $N - 2$. На Паскале и Бейсике мы уже умеем получать номер символа (функции Ord и ASC соответственно). На языке Си такой проблемы нет вообще, поскольку в нем символы можно обрабатывать как числовые значения, равные их кодам (номерам).

Итак, мы можем вычислить $N - 2$. Теперь остается только по значению $N - 2$ получить символ с соответствующим номером в таблице ASCII.

Для этой цели в Паскале служит функция Chr, обратная к Ord. В Бейсике существует аналогичная функция Chr$.

Дешифровка буквы будет выглядеть следующим образом:

Паскаль:

```
if Str[i] in ['C'.. 'Z', 'c'..'z'] then
  Str[i] := chr (ord (Str[i]) - 2);
```

Си:

```
if( ((Str[i] >= 'C') && (Str[i] <= 'Z')) ||
    ((Str[i] >= 'c') && (Str[i] <= 'z') )
  Str[i] = Str[i] - 2;
```

Бейсик:

```
Q$ = MID$ (Str$, i, 1)
IF ((Q$>="C") AND (Q$<="Z")) OR ((Q$>="c") AND (Q$<="z"))
THEN
Q$ = CHR$(ASC(Q$) - 2)
ENDIF
Str$=LEFT$(Str$,i-1)+Q$+RIGHT$(LEN(Str$)-i)
REM  Замена символа в строке
```

В Бейсике для замены символа в строке нам пришлось немного потрудиться, воспользовавшись функциями Left и Right, "вырезающими" заданное количество символов слева и справа строки соответственно. Впрочем, знатоки Бейсика могут предложить и другое решение с использованием оператора (не функции!) MID$:

```
MID$ (Str$, i, 1) = Q$
```

Для того чтобы решить проблему перехода через начало алфавита, случай первых букв можно рассмотреть отдельно, например

```
if Str[i] in ['C'..'Z', 'c'..'z'] then
  Str[i] = chr (ord (Str[i]) -2)
else
  if  Str[i] = 'B' then
    Str[i] := 'Z'
  else
    if Str[i] = 'A'  then
      Str[i] := 'Y'
```

Такое решение имеет право на существование, но лучше найти общую формулу. Смещение дешифрованной буквы от конца алфавита для прописных букв 'A' и 'B' определяется выражением `ord(Str[i])-2-ord('A')+1`. Покажем это. `ord(Str[i])-2` код дешифрованной буквы "вылезающий" за начало алфавита, в том случае, если он меньше `ord('A')`. Поэтому разность `(ord(Str[i])-2)-ord('A')` показывает, на сколько единиц код новый буквы "вылезает" за пределы алфавита. Если эта разность больше нуля, то имеет место переход через начало алфавита, а именно – если она 1, то новая буква должна быть 'Z', если 2, то 'Y' и т.д. Поэтому код дешифрованной буквы будет:

```
ord('Z') + ord (Str[i]) -2 - ord ('A') + 1.
```

Если бы смещение при дешифровке было бы равно не 2, а некоторому числу X (0<X<26), то формула для общего случая была бы:

```
ord('Z')+ord(Str[i])-X-ord ('A')+1.
```

Поскольку в английском алфавите 26 букв, то `ord('Z')-ord('A')+1 = 26`, то формулу можно упростить: `ord(Str[i])+26-X` в общем случае, и `ord(Str[i])+24` – в случае нашей задачи.

Приведем пример полного решения задачи:

Паскаль:

```
var c : char;
begin
 read (c); {читаем самый первый символ}
 while c <> '.' do
 begin
   {Обработка символа}
   if c in ['C'..'Z', 'c'..'z']  then
     c := chr (ord (c) -2)
   else
     if c in ['A'..'B', 'a'..'b']  then
       c := chr (ord (c) + 24);
   write (c)
   read (c); {читаем очередной символ}.
 end;
 writeln;
end.
```

Бейсик:

```
LINE INPUT Str$
i=1
WHILE NOT (MID$(Str$, i, 1) = ".")
REM  Обработка входной строки
Q$ = MID$ (Str$, i, 1)
IF ((Q$>="C") AND (Q$<="Z")) OR ((Q$>="c") AND (Q$<="z")) THEN
Q$ = CHR$(ASC(Q$) - 2)
ELSE
IF (Q$="A") OR ( Q$="a") OR ( Q$= "B") OR ( Q$ = "b") THEN
Q$=Chr$(ASC(Q$)+24)
ENDIF
Str=LEFT$(Str$,i-1)+Q$+RIGHT$(Len(Str$)-i)
REM  Замена символа в строке
i:=i+1
WEND
PRINT Str$
```

Си:

```
#include <stdio.h> /*включаем описания функций ввода-
вывода*/
int main(void) {
char Str [201] ;
int i = 0;
gets( Str); /* аргумент -  адрес массива Str*/
while (Str[i] != '.'){
```

```
    /*  Обработка входной строки*/
    if( ((Str[i] >= 'C') && (Str[i] <= 'Z')) ||
        ((Str[i] >= 'c') && (Str[i] <= 'z')) )
      Str[i] = Str[i] - 2;
    else
      if ((Str[i] == 'A') || (Str[i] == 'a') ||
          (Str[i] == 'B') || (Str[i] == '')||)
        Str[i] = Str[i] + 24;
    i++;
  }
  printf ("%s\n",  Str);
  return 0;
  }
```

Мы разобрали решение упрощенного задания ЕГЭ. В реальном задании величина сдвига не фиксирована, а равна минимальной длине слова в тексте, что делает задание более сложным. Вот текст реального задания.

На вход программе подается текст заклинания, состоящего не более чем из 200 символов, заканчивающийся точкой (символ "точка" во входных данных единственный). Оно было зашифровано Гарри Поттером следующим образом. Сначала Гарри определил количество букв в самом коротком слове, обозначив полученное число *K* (словом называется непрерывная последовательность английских букв, слова друг от друга отделяются любыми другими символами, длина слова не превышает 20 символов). Затем он заменил каждую английскую букву в заклинании на букву, стоящую в алфавите на *K* букв ранее (алфавит считается циклическим, то есть перед буквой *A* стоит буква *Z*), оставив другие символы неизменными. Строчные буквы при этом остались строчными, а прописные – прописными. Требуется написать программу на любом языке программирования, которая будет выводить на экран текст расшифрованного заклинания. Например, если зашифрованный текст был таким:

Zb Ra Ca Dab Ra.

то результат расшифровки должен быть следующим:

Bd Tc Ec Fcd Tc.

Предлагаем самостоятельно разобрать авторское решение задачи и критерии оценивания с учетом того, что задание отличается от рассмотренного нами упрощенного варианта еще и направлением сдвига при шифровке/дешифровке.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа читает входные данные, сразу подсчитывая минимальную длину встречающихся слов. За второй проход исходных данных производится замена букв латинского алфавита и печать расшифрованного сообщения. Баллы начисляются только за программу, которая решает задачу хотя бы для частного случая (например, для строчных английских букв и без циклического сдвига).

Пример правильной и эффективной программы на языке Паскаль:

```
var f:boolean;
    i, k, min: integer;
    c,cnew:char;
    s:string;
begin
  s:='';
  min:=250; k:=0;
  f:=false;
  repeat
    read(c);
    s:=s+c;
    if f then {слово началось}
      if c in ['a'..'z','A'..'Z'] then
        k:=k+1
      else
      begin
        if k<min then
          min:=k;
        f:=false
      end
    else {f=false}
      if c in ['a'..'z','A'..'Z'] then
      begin
        f:=true;
        k:=1
      end
   until c='.';
   for i:=1 to length(s) do
   begin
     cnew:=chr(ord(s[i])+min);
     case s[i] of
     'a'..'z':if cnew>'z' then
                write(chr(ord(cnew)-26))
              else
                write(cnew);
     'A'..'Z':if cnew>'Z' then
                write(chr(ord(cnew)-26))
              else
                write(cnew);
        else
          write(s[i])
     end;
   end;
   readln
end.
```

Пример правильной программы на языке Бейсик:

```
DIM i, j, min, k, f, a(26) AS INTEGER
INPUT s$
i = 1
k = 0
min = 250
f = 0
WHILE NOT (MID$(s$, i, 1) = ".")
c$ = MID$(s$, i, 1)
IF f = 1 THEN
IF (c$ >= "A") AND (c$ <= "Z") OR
   (c$ >= "a") AND (c$ <= "z") THEN
k = k + 1
ELSE
IF k < min THEN
min = k
f = 0
ENDIF
ELSE
IF (c$ >= "A") AND (c$ <= "Z") OR
   (c$ >= "a") AND (c$ <= "z") THEN
f = 1: k = 1
ENDIF
ENDIF
i = i + 1
WEND
IF k < min THEN
min = k
ENDIF
FOR j = 1 TO i
cnew$ = CHR$(ASC(MID$(s$, j, 1)) + min)
IF (MID$(s$, j, 1) >= "a") AND (MID$(s$, j, 1) <= "z") THEN
IF cnew$ > "z" THEN
PRINT (CHR$(ASC(cnew$) - 26));
ELSE
PRINT cnew$;
ENDIF
ELSE
IF (MID$(s$, j, 1) >= "A") AND (MID$(s$, j, 1) <= "Z") THEN
IF cnew$ > "Z" THEN
PRINT (CHR$(ASC(cnew$) - 26));
ELSE
PRINT cnew$;
ENDIF
ELSE
PRINT MID$(s$, j, 1);
ENDIF
ENDIF
NEXT j
END
```

Указания по оцениванию	Баллы
Программа работает верно, т.е. правильно расшифровывает сообщение, не меняя регистра английских букв, не содержит вложенных циклов (один – по количеству букв во входных данных, второй – по латинским буквам), в тексте программы не анализируется каждая буква в отдельности (if c ='a' then, if c ='b' then …). Допускается наличие в тексте программы одной пунктуационной ошибки.	4
Программа работает верно, но может содержать вложенные циклы или обрабатывает каждую букву явным образом (26 или 52 оператора IF или оператор CASE, содержащий 26 или 52 вариантов). Допускается наличие от одной до трех различных синтаксических ошибок: пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования, не описана или неверно описана переменная, применяется операция, недопустимая для соответствующего типа данных.	3
Программа работает в целом верно, но неправильно вычисляет букву при необходимости зацикливания алфавита или сдвиг производится не в том направлении. Возможно, в реализации алгоритма содержатся 1–2 ошибки (используется знак "<" вместо ">", "or" вместо "and" и т.п.). Допускается наличие от одной до пяти различных синтаксических ошибок.	2
Программа неверно работает при некоторых входных данных (например, со строчными буквами или текстами, содержащими символы, отличные от латинских букв), возможно, содержит ошибку в алгоритме поиска минимальной длины слова, в результате которой не всегда правильно находится величина сдвига. Допускается наличие от одной до семи различных синтаксических ошибок.	1
Задание выполнено неверно	0
Максимальный балл	4

Рассмотрим решение и критерии оценивания еще одной задачи ЕГЭ, связанной с обработкой символьных данных:

На вход программе подается последовательность символов, среди которых могут быть и цифры, отличные от нуля. Ввод символов заканчивается точкой (в программе на языке Бейсик символы можно вводить по одному в строке, пока не будет введена точка). Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая составит из тех цифр, которые не встречаются во входных данных, минимальное число (ноль не используется). Каждая цифра при этом используется ровно один раз. Если во входных данных встречаются все цифры от 1 до 9, то следует вывести "0".

Например, пусть на вход подаются следующие символы:
1A734B39.
В данном случае программа должна вывести
2568

Ответ:

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа читает все входные символы до точки один раз, помечая в массиве, состоящем из 9 элементов, какие цифры встретились во входных данных. Сами цифры при этом не запоминаются. За дополнительный проход этого массива печатаются те цифры, которые оказались не помеченными, в противном случае выводится 0. Баллы начисляются только за программу, которая решает задачу хотя бы для одного частного случая (например, для строк, состоящих не более чем из 255 символов, или когда в результате получаются небольшие числа).

Пример правильной и эффективной программы на языке Паскаль:

```
var a: array[1..9] of boolean;
    c: char;
    i, k: integer;
begin
  for i:= 1 to 9 do
    a[i]:=false;
  read(c);
  while c<>'.' do
  begin
    k:=ord(c)-ord('0');
    if (k>0) and (k<10) then
      a[k]:=true;
    read(c);
  end;
  k:=0;
  for i:=1 to 9 do
    if not a[i] then
    begin
      k:=k+1;
      write(i)
    end;
  if k=0 then
    write(0);
  writeln
end.
```

Пример правильной и эффективной программы на языке Бейсик:

```
DIM i, j, k, a(9) AS INTEGER
FOR i = 1 TO 9
a(i) = 0
NEXT;
INPUT c$
DO WHILE NOT (c$ = ".")
j = ASC(c$)
j = j - ASC("0") + 1
IF (j>0) AND (j<10) THEN
a(j) = a(j) + 1
ENDIF
INPUT c$
LOOP
k = 0
FOR i = 1 TO 9
IF a(i) = 0 THEN
k = k + 1
PRINT CHR$(i + 48);
ENDIF
NEXT
IF k = 0 THEN
PRINT 0
ENDIF
END
```

Указания по оцениванию	**Баллы**
Программа работает верно для любых входных данных произвольного размера и строит решение, не сохраняя входные данные в строке или массиве символов. Программа просматривает входные данные один раз, в тексте программы не анализируется каждая цифра в отдельности. Допускается наличие в тексте программы одной пунктуационной ошибки.	4
Программа работает верно, но входные данные запоминаются в массиве символов или строке, или входные данные считываются несколько раз. Возможно, каждая цифра обрабатывается явным образом (9 операторов IF, в том числе с использованием многоточия при записи программы, или оператор CASE, содержащий 9 – 10 вариантов). Возможно, после сохранения входных данных для каждой цифры они просматриваются заново и анализируется наличие соответствующей цифры, или из отобранных цифр формируется число вместо печати этих цифр в порядке возрастания. В программе присутствуют вложенные циклы (один по входным данным, второй – по цифрам, он может быть заменен оператором CASE или 9 операторами IF). Допускается наличие от одной до трех синтакси-	3

Указания по оцениванию	Баллы
ческих 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.	
Программа работает в целом верно, эффективно или нет, но в реализации алгоритма содержатся 1 – 2 ошибки (выход за границу массива, перевод символов в числа, из отобранных цифр формируется число, которое не помещается в используемый тип данных, например, 16-битное целое и т.п.). Возможно, некорректно организовано считывание входных данных. Допускается наличие от одной до пяти 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.	2
Программа, возможно, неверно работает при некоторых входных данных. Возможно, программа не определяет или неверно определяет, что все цифры во входных данных встречались, или содержит другие ошибки в выводе ответа. Допускается до 4 различных ошибок в реализации алгоритма, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи 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.	1
Задание не выполнено или выполнено неверно	0
Максимальный балл	4

Для закрепления изученного материала также рекомендуется выполнить следующие упражнения:

На вход программе подается текстовая строка (латинские буквы, цифры и разделители) длиной не более 200 символов, заканчивающаяся точкой (других точек в строке нет). К разделителям относятся: пробел, табуляция, запятая, восклицательный знак, двоеточие, точка с запятой.

а) Написать программу, подсчитывающую количество слов в исходной строке. Словом считается непустая последовательность символов, не содержащая разделителей.

Пример:
Исходные данные: **jhjsaRTYshfjk djDfhjk !:;a:::123a,,,2b:::b.**
Правильный результат: **6**

б) Написать программу, подсчитывающую в исходной строке количество слов, являющихся целыми десятичными числами.

Пример:
Исходные данные: **jhjdshfjk djGHjk !:;a::00:1,23,a,,,2b:::7.**
Правильный результат: **4** (00 1 23 7)

в) Написать программу, подсчитывающую в исходной строке количество слов, являющихся палиндромами, т.е. словами, читающимися одинаково справа налево и слева направо.

Пример:
Исходные данные: **jhjdshfjk djGHjk !:;a::00:1,232,as,,,2b:::7.**
Правильный результат: **5** (a 00 1 232 7)

г) Написать программу, меняющую в словах исходной строки, являющихся целыми числами, порядок цифр на обратный.

Пример:
Исходные данные: **jhjdshfjk 12345 djGHjk !:;a::00:1,235,as,,,2b:::7.**
Правильный результат: **jhjdshfjk 54321 djGHjk !:;a::00:1,532,as,,,2b:::7.**

д) Написать программу, заменяющую в исходной строке слова, являющиеся целыми числами на сумму их цифр.

Пример:
Исходные данные: **jhjdshfjk 12345 djGHjk !:;a::00:1,235,as,,,2b:::7.**
Правильный результат: **jhjdshfjk 15 djGHjk !:;a::0:1,10,as,,,2b:::7.**

Алгоритмы, типичные для заданий С4

Задания С4 весьма многообразны, однако в их решениях можно выделить несколько достаточно часто встречающихся элементарных алгоритмов:

- поиск минимума (максимума), возможно, среди элементов входных данных, удовлетворяющих некоторому дополнительному условию;
- поиск среднего арифметического значения, возможно, среди элементов входных данных, удовлетворяющих некоторому дополнительному условию;
- поиск значения, ближайшего по величине к минимуму/максимуму (но не равного им), возможно, среди элементов входных данных, удовлетворяющих некоторому дополнительному условию;
- упорядочение (сортировка) элементов входных данных.

На алгоритмах поиска минимума/максимума и среднего арифметического подробно останавливаться не будем, поскольку соответствующие алгоритмы уже упоминались в этой книге (обсуждение заданий С2) и рассматриваются практически во всех курсах информатики.

Рассмотрим алгоритм поиска значения элемента массива, ближайшего по значению к минимальному, но не равного ему, в массиве, значения всех элементов которого различны. Для простоты предположим, что все элементы массива – целые числа.

Очевиден следующий алгоритм – сортируем массив по возрастанию любым способом и ищем элемент, следующий за минимальными (учитывая, что элементов с минимальными значениями может быть несколько). Этот алгоритм не является эффективным, поскольку он производит избыточные действия над массивом – сортировку. Более эффективный алгоритм заключается в следующем: за первый проход находим в массиве минимальный (-ые) элемент(-ы) и удаляем его (их) из массива или помечаем его (их) некоторым образом. За второй проход снова ищем минимальный элемент, но уже среди неудаленных (непомеченных). Этот алгоритм более эффективен, чем способ с сортировкой, но есть еще более эффективный алгоритм, позволяющий достичь цели за один проход по массиву.

Пример фрагмента программы на Паскале

```
const N = 100; {максимальное число элементов массива}
var A : array [1..N] of integer;
        min, min_min, i: integer;
begin
...
 min := A[1];
 min_min := A[2];
 for i:= 2 to N do
 begin
  if min>A[i] then {смена минимума и ближайшего к нему значения}
  begin
    min_min := min;
    min := A[i]
  end
```

```
  else
    if min_min > A[i] then
      min_min := A[i]
    {смена  ближайшего к минимуму значения }
end;
 writeln (min_min); {печать  результата}
end.
```

Пример фрагмента программы на Бейсике

```
N=100
DIM A(N) AS INTEGER
…
min = A(1)
min_min = A(2)
FOR i= 2 TO N
IF min > A(i) THEN 'смена минимума и ближайшего к нему знач
min_min = min
min  = A(i)
ELSE
IF min_min > A(i) THEN
min_min = A(i) 'смена только ближайшего к минимуму знач
ENDIF
ENDIF
NEXT i
PRINT min_min 'печать  результата
```

Пример фрагмента программы на Си

```
#define N 100 /*максимальное число элементов массива*/
int A  [100];
int   min, min_min, i ;
{
…
 min = A[1];
 min_min = A[2];
 for (i= 2; i< N; i++) {
   if  (min > A[i]) { /* смена минимума и ближайшего
                         к нему значения */
     min_min = min;
     min  = A[i];
   }
   else
     if (min_min > A[i])
       min_min = A[i] /* смена только ближайшего
                         к минимуму значения */
 }
 printf ("%d\n", min_min); /*печать  результата*/
}
```

Заметим, что если в массиве возможны элементы с одинаковыми значениями, то задача несколько усложняется. Предлагаем самостоятельно написать программу для этого случая.

Пример еще одного задания С4 ЕГЭ на "бензиновую" тему, решение которого связано с поиском ближайшего значения к минимуму.

На автозаправочных станциях (АЗС) продается бензин с маркировкой 92, 95 и 98. В городе М был проведен мониторинг цены бензина на различных АЗС.

Напишите эффективную, в том числе и по используемой памяти, программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет определять для бензина с маркировкой 92, на какой АЗС его продают по второй по минимальности цене (считается, что самой низкой цене потребители не доверяют), а если таких АЗС несколько, то выдается только количество таких АЗС. Если все АЗС, у которых 92-й бензин есть, продают его по одной и той же цене, то эта цена считается искомой и выдается либо число таких АЗС, когда их несколько, либо конкретная АЗС, если она одна. Гарантируется, что хотя бы одна АЗС 92-й бензин продает.

На вход программе сначала подается число данных о стоимости бензина *N*. В каждой из следующих *N* строк находится информация в следующем формате:

<Компания> <Улица> <Марка> <Цена>

где <Компания> – строка, состоящая не более, чем из 20 символов без пробелов,
<Улица> – строка, состоящая не более, чем из 20 символов без пробелов,
<Марка> – одно из чисел – 92, 95 или 98,
<Цена> – целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках.

<Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <цена> разделены ровно одним пробелом. Пример входной строки:

МигОйл Мичуринский 92 1950

Программа должна выводить через пробел Компанию и Улицу искомой АЗС или их количество, если искомых вариантов несколько. Пример выходных данных:

ТНК Можайский

Второй вариант выходных данных:

4

Ответ:

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа читает все входные данные один раз, не запоминая их в массиве размер которого соответствует числу входных данных *N* или максимальной цене (3000). Во время чтения данных определяются две минимальных цены и количество АЗС, продающих по 92-й бензин по этим ценам. При печати результата проверяется, что у кого-то цена больше минимальной (вторая по минимальности цена существует), в этом случае искомая (искомые) АЗС – со второй по величине ценой, если это не так, то искомая (искомые) АЗС – все, продающие 92-й бензин.

Содержание верного ответа и указания по оцениванию
(допускаются иные формулировки ответа, не искажающие его смысла)

Баллы начисляются только за программу, которая решает задачу хотя бы для одного частного случая (например, когда все АЗС продают бензин по различной цене, и 92-й бензин продают не менее двух АЗС).

Ниже приведены примеры решения задания на языках Бейсик и Паскаль. Допускаются решения, записанные на других языках программирования. При оценивании решений на других языках программирования необходимо учитывать особенности этих языков программирования.

Пример правильной и эффективной программы на языке Паскаль:

```
var c: char;
    i, k, N, b, min1, min2, cnt1, cnt2: integer;
    s,s1,s2: string;
begin
  min1:=3001;
  cnt1:=0;
  readln(N);
  for i:=1 to N do
  begin
    read(c);
    s:='';
    repeat
      s:=s+c;
      read(c);
    until c=' '; {считана компания}
    repeat
      s:=s+c;
      read(c);
    until c=' '; {улица добавлена к компании}
    readln(k,b);
    if k = 92 then
    if min1 > b then
      begin
        min2:=min1; cnt2:=cnt1; s2:=s1;
        min1:=b; cnt1:=1; s1:=s
      end else
    if min1 = b then cnt1:=cnt1+1 else
    if min2 > b then
      begin
        min2:=b; cnt2:=1; s2:=s
      end else
    if min2 = b then cnt2:=cnt2+1
  end;
  if cnt2>0 then
    if cnt2=1 then writeln(s2) else writeln(cnt2)
  else {все АЗС продают 92-й бензин по одной цене}
    if cnt1=1 then writeln(s1) else writeln(cnt1);
  writeln;
end.
```

Пример правильной программы на языке Бейсик:

```
min1 = 3001
cnt1 = 0
INPUT n
FOR j = 1 TO n
LINE INPUT s$
i = 0
DO
i = i + 1
c$ = MID$(s$, i, 1)
LOOP WHILE c$ <> " "
DO
i = i + 1
c$ = MID$(s$, i, 1)
LOOP WHILE c$ <> " "
DO
i = i + 1
c$ = MID$(s$, i, 1)
LOOP WHILE c$ <> " "
m = VAL(MID$(s$, i + 1, 2))
b = VAL(MID$(s$, i + 4))
k = i - 1
s = LEFT$(s$, k)
IF m = 92 THEN
IF min1 > b THEN
min2 = min1: cnt2 = cnt1: s2$ = s1$
min1 = b: cnt1 = 1: s1$ = s$
ELSE
IF min1 = b THEN
cnt1 = cnt1 + 1
ELSE
IF min2 > b THEN
min2 = b: cnt2 = 1: s2$ = s$
ELSE
IF min2 = b THEN cnt2 = cnt2 + 1
ENDIF
ENDIF
ENDIF
ENDIF
NEXT j
IF cnt2 > 0 THEN
IF cnt2 = 1 THEN PRINT s2$ ELSE PRINT cnt2
ELSE
IF cnt1 = 1 THEN PRINT s1$ ELSE PRINT cnt1
ENDIF
END
```

Указания по оцениванию	Баллы
Программа работает верно для любых входных данных произвольного размера и находит ответ, не сохраняя входные данные в массиве. Программа просматривает входные данные один раз, используя для нахождения ответа два набора переменных (значение минимума, количество таких элементов и компания+улица для одной АЗС) для минимальной и второй по величине цены. Допускается наличие в тексте программы одной синтаксической ошибки.	4
Программа работает верно, но входные данные запоминаются в массиве, в том числе возможно в массиве с индексами от 0 до 3000, обозначающем количество АЗС, продающих 92-й бензин по соответствующей цене, или входные данные считываются несколько раз. Возможно вместо алгоритма поиска двух минимумов используется сортировка цен. Допускается наличие от одной до трех 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.	3
Программа работает в целом верно, эффективно или нет, но, в реализации алгоритма содержатся 1–2 ошибки (выход за границу массива, перевод символов в числа, используется знак "<" вместо "<=", "or" вместо "and" и т.п.). Одной из двух ошибок может быть следующая логическая ошибка: при печати результата не проверяется, есть ли АЗС с ценой отличной от минимальной, и всегда печатается информация по второму минимуму. Возможно, некорректно организовано считывание входных данных. Допускается наличие от одной до пяти синтаксических ошибок, описанных выше.	2
Программа, возможно не всегда верно определяет второй минимум (например, текущий элемент сравнивается только с текущим минимумом и не сравнивается со вторым минимумом) или искомое количество АЗС. При использовании сортировки допущены ошибки в ее реализации. Допускается до 4 различных ошибок в реализации алгоритма, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи синтаксических ошибок, описанных выше.	1
Задание не выполнено или выполнено неверно	0
Максимальный балл	*4*

Один из наиболее простых алгоритмов сортировки – сортировка простыми обменами (пузырьковая сортировка) – уже был рассмотрен при анализе заданий С2.

На вход программе подаются сведения о пассажирах, сдавших свой багаж в камеру хранения. В первой строке задано текущее время: через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа). Во второй строке сообщается количество пассажиров N, которое не меньше 10, но не превосходит 1000. Каждая из следующих N строк имеет следующий формат:

<Фамилия> <время освобождения ячейки>,

где <Фамилия> – строка, состоящая не более, чем из 20 символов,

<время освобождения ячейки> – через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа).

<Фамилия> и <время освобождения ячейки> разделены одним пробелом. Сведения отсортированы в порядке времени сдачи багажа.

Требуется написать на любом языке программирования программу, выводящую фамилии пассажиров, которые в ближайшие 2 часа должны освободить ячейки, в хронологическом порядке освобождения ячеек.

Пример входных данных:

10:00

3

Иванов 12:00

Петров 10:00

Сидоров 12:12

Результат работы программы для этого примера

Петров

Иванов

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа верно читает входные данные, сразу запоминая в массиве только фамилии и времена окончания хранения багажа тех пассажиров, которые должны освободить ячейки в ближайшие 2 часа. Время при считывании удобно перевести в минуты и в этом же виде хранить и сравнивать. Затем полученный массив времен сортируется по неубыванию любым алгоритмом сортировки, параллельно переставляются и элементы массива с фамилиями (возможно использование одного массива записей, состоящих из двух полей). Печатаются элементы массива фамилий в полученном в результате сортировки порядке.

Пример правильной и эффективной программы на языке Паскаль:

```
type pp=record
          name:string[20];
          time:integer;
        end;
var
  p:array[1..1000]of pp;
  q:pp;
  c,c1:char;
  i,j,N,time1:integer;
begin
  read(c,c1); {считаны часы текущего времени}
  time1:=60*((ord(c)-ord('0'))*10+ ord(c1)-ord('0'));
  readln(c,c,c1); {пропущено двоеточие, и считаны минуты}
  time1:=time1+(ord(c)-ord('0'))*10+ord(c1)-ord('0');
  readln(N);
  j:=1;
  for i:=1 to N do
  begin
    p[j].name:='';
    repeat
      read(c);
      p[j].name:=p[j].name+c
    until c=' '; {считана фамилия}
    read(c,c1); {считаны часы первого времени}
    p[j].time:=60*((ord(c)-ord('0'))*10+ ord(c1)-ord('0'));
    readln(c,c,c1); {пропущено двоеточие, и считаны минуты}
    p[j].time:=p[j].time+
               (ord(c)-ord('0'))*10+
                ord(c1)-ord('0');
    if (p[j].time>=time1) and
       (p[j].time<=time1+120)then
      j:=j+1; {данные занесены в массив}
  end;
  N:=j-1;
  for i:=1 to N-1 do {сортируем данные}
    for j:=1 to N-i do
      if p[j].time>p[j+1].time then
        begin
          q:=p[j];
          p[j]:=p[j+1];
          p[j+1]:=q;
        end;
  for i:=1 to N do
    writeln(p[i].name)
end.
```

Пример правильной программы на языке Бейсик:

```
  DIM t(1000) AS INTEGER
  DIM m(1000) AS STRING * 20
  LINE INPUT s$
  time1 = (ASC(MID$(s$, 1, 1)) - ASC("0")) * 60 * 10
  time1 = time1 + (ASC(MID$(s$, 2, 1)) - ASC("0")) * 60
  time1 = time1 + (ASC(MID$(s$, 4, 1)) - ASC("0")) * 10
  time1 = time1 + (ASC(MID$(s$, 5, 1)) - ASC("0"))
  INPUT N
  k = 0
  FOR j = 1 TO N
  LINE INPUT s$
  c$ = MID$(s$, 1, 1)
  i = 1
  WHILE NOT (c$ = " ")
   i = i + 1
   c$ = MID$(s$, i, 1)
  WEND
  nm$ = MID$(s$, 1, i)
  time2 = (ASC(MID$(s$, i + 1, 1)) - ASC("0")) * 60 * 10
  time2 = time2 + (ASC(MID$(s$, i + 2, 1)) - ASC("0")) * 60
  time2 = time2 + (ASC(MID$(s$, i + 4, 1)) - ASC("0")) * 10
  time2 = time2 + (ASC(MID$(s$, i + 5, 1)) - ASC("0"))
  IF time2 >= time1 AND time2 <= time1 + 120 THEN
  k = k + 1
  t(k) = time2
  m$(k) = nm$
  ENDIF
  NEXT j
  FOR i = 1 TO k - 1
  FOR j = 1 TO k - i
  IF t(j) > t(j + 1) THEN
  time2 = t(j): nm$ = m$(j)
  t(j) = t(j + 1): m$(j) = m$(j + 1)
  t(j + 1) = time2: m$(j + 1) = nm$
  ENDIF
  NEXT j
  NEXT i
  FOR i = 1 TO k
  PRINT m$(i)
  NEXT i
END
```

Указания по оцениванию	Баллы
Программа работает верно и эффективно, т.е. корректно выделяет из входных данных время, запоминает фамилии пассажиров и время выдачи багажа только тех пассажиров, которые должны забрать его в ближайшие 2 часа. Фамилии этих пассажиров верно сортируются согласно временам выдачи багажа, а затем печатаются. Допускается наличие в тексте программы одной синтаксической ошибки.	4
Программа работает в целом верно, но содержит по крайней мере две из следующих нерациональностей: сохраняются фамилии и времена для всех пассажиров, время не переводится в минуты, сортируются все фамилии, а при печати анализируется, какие из них допустимые. Допускается наличие от одной до трех 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.	3
Программа работает в целом верно, но не всегда верно определяет допустимость времени или некорректно работает в случае отсутствия допустимых времен. Возможно, в реализации алгоритма содержатся 1–2 ошибки (используется знак «<« вместо «>«, «or» вместо «and» и т.п.). Возможно, некорректно организовано считывание входных данных. Допускается наличие до пяти 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.	2
Программа неверно работает при некоторых входных данных и, возможно, содержит ошибку в сортировке, или времена сортируются верно, а соответствующие им фамилии – нет. Допускается до 4 различных ошибок в ходе решения задачи, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи 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.	1
Задание выполнено неверно	0
Максимальный балл	*4*

Приведем несколько заданий ЕГЭ для самостоятельной работы. Учитывая, что разбирать чужую программу обычно труднее, чем писать собственную, предлагается следующий алгоритм действий:

- разработать свой алгоритм решения и написать программу на бумаге, в случае серьезных затруднений воспользоваться подсказками из авторского решения;
- если есть возможность – отладить программу на компьютере, исправив ошибки в бумажном варианте, и оценить по критериям первоначальный бумажный вариант;
- сравнить свой вариант программы с авторским;

4.1. На вход программе подаются сведения об учениках некоторой средней школы. В первой строке сообщается количество учеников N, каждая из следующих N строк имеет следующий формат:

<Фамилия> <Имя> <класс>,

где <Фамилия> – строка, состоящая не более, чем из 20 символов,
<Имя> – строка, состоящая не более, чем из 15 символов,
<класс> – год обучения (от 1 до 12) и заглавная буква (от «А» до «Я») без пробела.

<Фамилия> и <Имя>, а также <Имя> <класс> разделены одним пробелом. Пример входной строки:

Иванов Петр 10Б

Требуется написать программу на любом языке программирования, которая будет выводить на экран информацию о параллелях (годе обучения) с наибольшим числом учеников. Программа должна выводить на экран в первой строке количество учеников в искомых параллелях, а во второй строке – в порядке возрастания номера этих параллелей через пробел. Например:

100
1 7 11

4.2. 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 4

Требуется написать программу на любом языке программирования, которая будет выводить на экран фамилии и имена трех лучших по среднему баллу учеников. Если среди остальных есть ученики, набравшие тот же средний балл, что и один из трех лучших, то следует вывести и их фамилии и имена.

4.3. Заключительный этап олимпиады по астрономии проводился для учеников 9-11-х классов, участвующих в общем конкурсе. Каждый участник олимпиады мог набрать от 0 до 50 баллов. Для определения победителей и призеров сначала отбираются 45% участников, показавших лучшие результаты.

По положению, в случае, когда у последнего участника, входящего в 45%, оказывается количество баллов такое же, как и у следующих за ним в итоговой таблице, решение по данному участнику и всем участникам, имеющим с ним равное количество

баллов, определяется следующим образом: все участники признаются призерами, если набранные ими баллы больше половины максимально возможных;

все участники не признаются призерами, если набранные ими баллы не превышают половины максимально возможных.

Напишите эффективную по времени работы и по используемой памяти программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая по результатам олимпиады будет определять, какой минимальный балл нужно было набрать, чтобы стать победителем или призером олимпиады.

На вход программе сначала подается число участников олимпиады *N*. В каждой из следующих *N* строк находится результат одного из участников олимпиады в следующем формате:

<Фамилия> <Имя> <класс> <баллы>

где <Фамилия> – строка, состоящая не более, чем из 20 символов,
<Имя> – строка, состоящая не более, чем из 15 символов,
<класс> – число от 9 до 11, <баллы> – целое число от 0 до 50 набранных участником баллов.

<Фамилия> и <Имя>, <Имя> и <класс>, а также <класс> и <баллы> разделены одним пробелом. Пример входной строки:

Иванов Петр 10 17

Программа должна выводить минимальный балл призера. Гарантируется, что хотя бы одного призера по указанным правилам определить можно.

ОТВЕТЫ И КРИТЕРИИ ОЦЕНИВАНИЯ

1.1. Решение уравнения $a/x=b$

```
Если a=0, то
  Если b=0, то
    Ответ "x – любое число, не равное 0"
  Иначе
    Ответ "нет решений"
Иначе
  Если b=0, то
    Ответ "нет решений"
  Иначе
    Ответ "x=a/b"
```

1.2. Решение уравнения $a/|x|=b$

```
Если a=0, то
  Если b=0, то
    Ответ "x – любое число, не равное 0"
  Иначе
    Ответ "нет решений"
Иначе
  Если b=0 или a/b<0, то
    Ответ "нет решений"
  Если a/b>0, то
    Ответ "x=a/b или x=–a/b"
```

1.3. Решение неравенства $a|x|>b$

```
Если a=0, то
  Если b<0, то
    Ответ "x – любое число"
  Иначе
    Ответ "нет решений"
Иначе
  Если a>0, то
    Если b/a>0, то
      Ответ "x>b/a или x<b/a"
    Иначе
      Ответ "x – любое число"
  Иначе
    Если b/a>0, то
      Ответ "–b/a<x<b/a"
    Иначе
      Ответ "нет решений"
```

1.4. Решение неравенства $a|x|\le b$

```
Если a=0, то
 Если b≥0, то
  Ответ "x – любое число"
 Иначе
  Ответ "нет решений"
Иначе
 Если a>0, то
  Если b/a>0, то
   Ответ "–b/a≤x≤b/a"
  Иначе
   Если b=0, то
    Ответ "x=0"
   Иначе
    Ответ "нет решений"
 Иначе
  Если b/a>0, то
   Ответ "x>b/a или x<–b/a"
  Иначе
   Ответ "x – любое число"
```

1.5. Решение неравенства $ax/(x-l$

```
Если a=0, то
 Ответ "x≠b"
Иначе
 Если a>0, то
  Если b=0, то
   Ответ "нет решений"
  Иначе
   Если b>0, то
    Ответ "0≤x<b"
   Иначе
    Ответ "b<x≤0"
 Иначе
  Если b=0, то
   Ответ "x>0 или x<0"
  Иначе
   Если b>0, то
    Ответ "x>b или x≤0"
   Иначе
    Ответ "x<b или x≥0"
```

1.6.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)	**Баллы**
Элементы ответа: 1) Пример: a= –1 b=0 x=1 (значение *x* может быть не указано) 2) Лишняя часть: не нужно вводить *x* с клавиатуры верно: `readln(a,b);` 3) Возможная доработка: `readln(a,b,x);` `if a=0 then` `  if -b<=0 then` `    write('любое число')` `  else` `    write('нет решений')` `else` `  if a>0 then` `    write('x>=',b/a)` `  else` `    write('x<=',b/a);` (могут быть и другие способы доработки). При оценке других вариантов доработки программы нужно проверять, что поставленная цель достигается.	
Указания по оцениванию	
Правильно выполнены все 3 пункта задания, при этом в работе (во фрагментах программ) допускается не более одной синтаксической ошибки	3
Правильно выполнены 2 пункта задания. При этом в сданной работе допускается не более двух синтаксических ошибок (пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования)	2
Правильно выполнен только один пункт задания, при этом, если это был п.3), то в нем допускается не более трех синтаксических ошибок (пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования)	1
Все пункты задания выполнены неверно	0
Максимальный балл	3

1.7.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)	**Баллы**
Элементы ответа: 1) Пример: a=1 b=–1 x=1 (значение *x* может быть не указано) 2) Лишняя часть: не нужно вводить *x* с клавиатуры верно: `readln(a,b);` 3) Возможная доработка: `readln(a,b,x);` `if a=0 then` `  write('x<>',b)` `else` `  if b=0 then` `  write('x<>0')` `  else` `  if b>0 then` `  write('x>=0 или x<',b)` `  else` `  write('x<=0 или x>',b)` (могут быть и другие способы доработки). При оценке других вариантов доработки программы нужно проверять, что поставленная цель достигается.	
Указания по оцениванию	
Правильно выполнены все 3 пункта задания, при этом в работе (во фрагментах программ) допускается не более одной синтаксической ошибки	3
Правильно выполнены 2 пункта задания. При этом в сданной работе допускается не более двух синтаксических ошибок (пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования)	2
Правильно выполнен только один пункт задания, при этом, если это был п.3), то в нем допускается не более трех синтаксических ошибок (пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования)	1
Все пункты задания выполнены неверно	0
Максимальный балл	3

1.8.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)	**Баллы**
Элементы ответа: 1) Пример: x=2π, y=0 (Любая пара (x,y), для которой выполняется: y>x–1 или y>0 или (y>= –sin x и y≤0 и x≥2π)) 2) Возможная доработка: `if (y<=x-1) and (y<=0) and (y>=-sin(x)) and (x<4) then` `  write('принадлежит')` `else` `  write('не принадлежит')` Могут быть и другие верные способы доработки.	
Указания по оцениванию	
Обратите внимание! В задаче требовалось выполнить **три** действия: указать пример входных данных, при которых программа работает неверно и исправить две ошибки: 1. Неправильное использование условного оператора, в результате чего при невыполнении первого, второго или третьего условия программа не выдавала ничего (отсутствуют случаи ELSE). 2. Приведенным трем ограничениям удовлетворяют также точки плоскости, у которых (y>= –sin x) и (y≤0) и (x≥2π).	
Правильно выполнены все **три** действия. Исправлены обе ошибки. В работе (во фрагментах программ) допускается наличие отдельных синтаксических ошибок, не искажающих замысла автора решения	3
Правильно выполнены два действия из трех (исправлены обе ошибки, но не указан/неправильно указан пример требуемых входных данных, либо правильно указан пример входных данных, программа правильно работает при большем числе случаев, чем исходная, но не при всех). Например, выдает "принадлежит" для точек, у которых (y>= –sin x) и (y≤0) и (x≥2π). При этом не допускается, чтобы программа неправильно работала при тех входных данных, при которых раньше работала правильно (даже если она при этом правильно стала работать при большем количестве входных данных, чем исходная). ИСКЛЮЧЕНИЕ! При написании операций сравнения допускается одно неправильное использование строгих/нестрогих неравенств (считается несущественной ошибкой, погрешностью записи). Например, вместо "y<=0" используется "y<0" (даже если программа при этом стала неверно работать при тех входных данных, при которых раньше работала правильно). Допускается, например, такое решение: `if y<=x-1 then` `  if y<=0 then` `  if y>= -sin(x) then` `  write('принадлежит')`	2

`` else write('не принадлежит') else write('не принадлежит') else write('не принадлежит') ```	
Правильно выполнено только одно действие из трех. То есть, либо только приведен пример входных данных, либо он не приведен (или приведен неверно), но имеется программа, корректно работающая при большем количестве входных данных, чем исходная, но не при всех (допускается применение исключения, описанного в критериях оценки задачи на 2 балла).	1
Все пункты задания выполнены неверно (пример входных данных не указан или указан неверно, программа не приведена, либо приведенная программа корректно работает в не большем количестве случаев, чем исходная).	0
Максимальный балл	3

1.9.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)	**Баллы**
Элементы ответа: 1) Пример: x=π, y=0 (Любая пара (x,y), для которой выполняется: x<0 или y>0,5 или (y>=sin x и y≤0,5 и x≥5π/6)) 2) Возможная доработка: `if (x>=0) and (y<=0.5) and (y>=sin(x)) and (x<1) then` `  write('принадлежит')` `else` `  write('не принадлежит')` Могут быть и другие верные способы доработки.	
Указания по оцениванию	
Обратите внимание! В задаче требовалось выполнить **три** действия: указать пример входных данных, при которых программа работает неверно и исправить две ошибки: 1. Неправильное использование условного оператора, в результате чего при невыполнении первого, второго или третьего условия программа не выдавала ничего (отсутствуют случаи ELSE). 2. Приведенным трем ограничениям удовлетворяют также точки плоскости, у которых (y>=sin x) и (y≤0,5) и (x≥5π/6).	
Правильно выполнены все **три** действия. Исправлены обе ошибки. В работе (во фрагментах программ) допускается наличие отдельных синтаксических ошибок, не искажающих замысла автора решения	3
Правильно выполнены два действия из трех (исправлены обе ошибки, но не указан/неправильно указан пример требуемых входных данных, либо правильно указан пример входных данных, программа правильно работает при большем числе случаев, чем исходная, но не при всех). Например, выдает "принадлежит" для точек, у которых (y>=sin x) и (y≤0,5) и (x≥5π/6). При этом не допускается, чтобы программа неправильно работала при тех входных данных, при которых раньше работала правильно (даже если она при этом правильно стала работать при большем количестве входных данных, чем исходная). ИСКЛЮЧЕНИЕ! При написании операций сравнения допускается одно неправильное использование строгих/нестрогих неравенств (считается несущественной ошибкой, погрешностью записи). Например, вместо "x>=0" используется "x>0" (даже если программа при этом стала неверно работать при тех входных данных, при которых раньше работала правильно). Допускается, например, такое решение: `IF x>=0 THEN` `  IF y<=0.5 THEN` `   IF y>= SIN(x) THEN` `      write('принадлежит')`	2

`else` `  write('не принадлежит')` `else` `  write('не принадлежит')` `else` `  write('не принадлежит')`	
Правильно выполнено только одно действие из трех. То есть, либо только приведен пример входных данных, либо он не приведен (или приведен неверно), но имеется программа, корректно работающая при большем количестве входных данных, чем исходная, но не при всех (допускается применение исключения, описанного в критериях оценки задачи на 2 балла).	1
Все пункты задания выполнены неверно (пример входных данных не указан или указан неверно, программа не приведена, либо приведенная программа корректно работает в не большем количестве случаев, чем исходная).	0
Максимальный балл	3

Мы не стали приводить ответы на задания раздела С2, оставив их вам на самостоятельное решение.

Для всех заданий С3 критерии оценивания одинаковы:

Указания по оцениванию	Баллы
Правильное указание выигрывающего игрока и его ходов со строгим доказательством правильности (с помощью или без помощи дерева игры).	3
Правильное указание выигрывающего игрока, стратегии игры, приводящей к победе, но при отсутствии доказательства ее правильности.	2
При наличии в представленном решении одного из пунктов: 1. Правильно указаны все варианты хода первого игрока и возможные ответы второго игрока (в том числе и все выигрышные), но неверно определены дальнейшие действия и неправильно указан победитель. 2. Правильно указан выигрывающий игрок, но описание выигрышной стратегии неполно и рассмотрены несколько (больше одного, но не все) вариантов хода первого игрока и частные случаи ответов второго игрока.	1
Задание не выполнено или в представленном решении полностью отсутствует описание элементов выигрышной стратегии, и отсутствует анализ вариантов первого-второго ходов играющих (даже при наличии правильного указания выигрывающего игрока).	0
Максимальный балл	3

3.1. Выигрывает второй игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход
Стартовая позиция	I-й игрок (все варианты хода)	II-й игрок (выигрышный ход)	I-й игрок (все варианты хода)	II-й игрок (выигрышный ход)
3,2	6,2	6,6	9,6	12,6
			6,9	6,12
			6,10	6,13
	3,5	3,8	6,8	6,12
			3,11	3,14
			3,12	3,15
	3,6	6,6	9,6	12,6
			6,9	6,12
			6,10	6,13

Таблица содержит *все возможные* варианты ходов первого игрока. Из нее видно, что при любом ходе первого игрока у второго имеется ход, приводящий к победе.

3.2. Выигрывает первый игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
1,2	4,2	4,4	7,4	10,4	15,4
				12,4	15,4
				7,8	7,16
		7,2	7,4	10,4	15,4
				12,4	15,4
				7,8	7,16
		9,2	14,2	–	

Таблица содержит *все возможные* варианты ответных ходов второго игрока на выигрышный ход первого игрока. Из нее видно, что при любом ходе второго игрока у первого имеется ход, приводящий к победе.

3.3. Выигрывает первый игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

В данной задаче есть два первых хода, приводящих к победе. Вы можете привести любой из них. Первый вариант таблицы:

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
0,3	2,3	4,3	8,3	10,3	14,3
				12,3	14,3
				8,6	12,6
		6,3	8,3	10,3	14,3
				12,3	14,3
				8,6	12,6
		2,6	6,6	8,6	12,6
				10,6	12,6
				6,9	10,9

Второй вариант таблицы:

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
0,3	0,6	2,6	6,6	8,6	12,6
				10,6	12,6
				6,9	10,9
		4,6	6,6	8,6	12,6
				10,6	12,6
				6,9	10,9
		0,9	4,9	6,9	10,9
				8,9	10,9
				4,12	6,12

Таблица содержит *все возможные* варианты ответных ходов второго игрока на выигрышный ход первого игрока. Из нее видно, что при любом ходе второго игрока у первого имеется ход, приводящий к победе.

3.4. Выигрывает второй игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

	1 ход	2 ход	3 ход	4 ход	5 ход	6 ход
Стартовая позиция	I-й игрок (все варианты хода)	II-й игрок (выигрышный ход)	I-й игрок (все варианты хода)	II-й игрок (выигрышный ход)	I-й игрок (все варианты хода)	II-й игрок (выигрышный ход)
2,2	5,2	5,4	8,4	8,6	11,6	11,8
					8,8	11,8
					8,10	11,10
			5,6	8,6	11,6	11,8
					8,8	11,8
					8,10	11,10
			5,8	5,12	–	
	2,4	5,4	8,4	8,6	11,6	11,8
					8,8	11,8
					8,10	11,10
			5,6	8,6	11,6	11,8
					8,8	11,8
					8,10	11,10
			5,8	5,12	–	
	2,6	2,8	5,8	5,12	–	
			2,10	2,14	–	
			2,12	2,14	–	

Таблица содержит *все возможные* варианты ходов первого игрока. Из нее видно, что при любом ходе первого игрока у второго имеется ход, приводящий к победе.

3.5. Выигрывает первый игрок.

Для доказательства рассмотрим неполное дерево игры, оформленное в виде таблицы, где в каждой ячейке записаны координаты фишки на каждом этапе игры.

В данной задаче первый игрок может сделать любой первый ход и все равно выиграть. Вы можете привести любой из них.

Первый вариант таблицы:

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
2,0	5,0	8,0	8,4	11,4	11,8
				8,8	11,8
				10,6	12,8
		5,4	8,4	11,4	11,8
				8,8	11,8
				10,6	12,8
		7,2	9,4	12,4	12,8
				9,8	12,8
				11,6	13,8

Второй вариант таблицы:

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
2,0	2,4	5,4	5,8	8,8	8,12
				5,12	8,12
				7,10	9,12
		2,8	5,8	8,8	8,12
				5,12	8,12
				7,10	9,12
		4,6	7,6	10,6	10,10
				7,10	10,10
				9,8	11,10

Третий вариант таблицы:

	1 ход	2 ход	3 ход	4 ход	5 ход
Стартовая позиция	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)	II-й игрок (все варианты хода)	I-й игрок (выигрышный ход)
2,0	4,2	7,2	7,6	10,6	10,10
				7,10	10,10
				9,8	11,10
		4,6	7,6	10,6	10,10
				7,10	10,10
				9,8	11,10
		6,4	9,4	12,4	12,8
				9,8	12,8
				11,6	13,8

Таблица содержит *все возможные* варианты ответных ходов второго игрока на выигрышный ход первого игрока. Из нее видно, что при любом ходе второго игрока у первого имеется ход, приводящий к победе.

4.1.

Содержание верного ответа (допускаются иные формулировки ответа, не искажающие его смысла)
Программа верно читает входные данные, не запоминая их все, а сразу подсчитывая в массиве, хранящем 12 целых чисел, количество учащихся в каждой из параллелей. Затем с использованием этого массива ищется параллель с максимальным числом учеников. За дополнительный просмотр этого массива распечатывается информация об искомых параллелях. Баллы начисляются только за программу, которая решает задачу хотя бы для частного случая (например, параллель с максимальным количеством учеников единственна).

Пример правильной и эффективной программы на языке Паскаль:

```
var pc:array[1..12] of integer;
    p:1..12;
    class:string[3];
    c:char;
    max, i, N:integer;
begin
  readln(N);
  for i:=1 to 12 do
    pc[i]:=0;
  for i:=1 to N do
  begin
    repeat
      read(c)
    until c=' '; {считана фамилия}
    repeat
      read(c)
    until c=' '; {считано имя}
    readln(class);
    {определяем номер параллели}
    if length(class)=2 then
      p:=ord(class[1])-ord('0') else
     p:=(ord(class[1])-ord('0'))*10+
         ord(class[2])-ord('0');
    pc[p]:=pc[p]+1;{учитываем ученика этой параллели}
  end;
  max:=0;
  for i:=1 to 12 do
    if pc[i]>max then max:=pc[i];
  writeln('Максимум учеников в параллели:',max);
  for i:=1 to 12 do
    if pc[i]=max then
      write(i,' ');
  readln
end.
```

Пример правильной программы на языке Бейсик:

```
DIM i, j, p, n, max, pc(12) AS INTEGER
DIM m(12)
FOR i = 1 TO 12
    pc(i) = 0
NEXT i
INPUT n
FOR j = 1 TO n
LINE INPUT s$
c$ = MID$(s$, 1, 1)
i = 1
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s$, i, 1)
WEND
i = i + 1
c$ = MID$(s$, i, 1)
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s$, i, 1)
WEND
s = MID$(s$, i + 1, 3)
IF MID$(s$, 2, 1) >= "0" AND MID$(s$, 2, 1) <= "2" THEN
 p = (ASC(MID$(s$, 1, 1)) - ASC("0")) * 10 +
        ASC(MID$(s, 2, 1)) - ASC("0")
ELSE
 p = ASC(MID$(s$, 1, 1)) - ASC("0")
END IF
pc(p) = pc(p) + 1
NEXT j
max = 0
FOR i = 1 TO 12
  IF pc(i) > max THEN max = pc(i)
NEXT i
PRINT "Max = "; max
FOR i = 1 TO 12
  IF pc(i) = max THEN PRINT i; " ";
NEXT i
END
```

Указания по оцениванию	Баллы
Программа работает верно, т.е. корректно выделяет из входных данных номер параллели, не содержит вложенных циклов (от 1 до N и от 1 до 12 или от 1 до 26), в тексте программы не анализируется каждая параллель в отдельности. Допускается наличие в тексте программы одной пунктуационной ошибки.	4
Программа работает верно, но содержит вложенные циклы (от 1 до 12 и от 1 до N) или обрабатывает каждую параллель явным образом (12 операторов IF или оператор CASE, содержащий 12 вариантов). Возможно, сохраняет все входные данные в массиве учеников. Допускается наличие от одной до трех синтаксических ошибок: пропущен или неверно указан знак пунктуации, неверно написано зарезервированное слово языка программирования, не описана или неверно описана переменная, применяется операция, недопустимая для соответствующего типа данных.	3
Программа работает в целом верно, но выводит значение только одной параллели с максимальным числом учащихся. Возможно, в реализации алгоритма содержатся 1–2 ошибки (используется знак "<" вместо ">", "or" вместо "and" и т.п.). Допускается наличие до пяти синтаксических ошибок.	2
Программа неверно работает при некоторых входных данных и, возможно, содержит ошибку в алгоритме поиска максимума или в выделении номера параллели из строки входных данных. Допускается наличие до семи синтаксических ошибок.	1
Задание выполнено неверно	0
Максимальный балл	4

4.2.

Содержание верного ответа и указания по оцениванию
Программа верно читает входные данные, запоминая фамилии, имена и сумму баллов в массиве записей (или в нескольких массивах), сразу или за дополнительный просмотр подсчитывая три лучших по величине суммы баллов (так как количество экзаменов у всех учеников одинаковое, лучший средний балл соответствует лучшей сумме баллов). Затем за дополнительный просмотр этого массива распечатывается информация о тех учениках, которые набрали в сумме баллов не меньше третьей по величине суммы. Баллы начисляются только за программу, которая решает задачу хотя бы для частного случая (например, все ученики набрали различный средний балл).

Пример правильной и эффективной программы на языке Паскаль:

```
var p:array[1..100] of record
                         name:string;
                         sum:integer;
                       end;
    c:char;
    i,j,N,s1,s2,s3,m:integer;
begin
  readln(N);
  for i:=1 to N do
  begin
    p[i].name:='';
    repeat
      read(c);
      p[i].name:=p[i].name+c
    until c=' '; {считана фамилия}
    repeat
      read(c);
      p[i].name:=p[i].name+c
    until c=' '; {считано имя}
    p[i].sum:=0;
    for j:=1 to 3 do
    begin
      read(m);
      p[i].sum:=p[i].sum+m
    end; {подсчитана сумма баллов}
    readln;
end;
s1:=0; s2:=0; s3:=0;
  for i:=1 to N do
  begin
    if p[i].sum>s1 then
      begin
        s3:=s2; s2:=s1;
        s1:=p[i].sum
      end else
    if p[i].sum>s2 then
      begin
        s3:=s2; s2:=p[i].sum
      end else
    if p[i].sum>s3 then s3:=p[i].sum;
  end;
  for i:=1 to N do
    if p[i].sum>=s3 then writeln(p[i].name);
end.
```

Пример правильной программы на языке Бейсик:

```
DIM i, j, n, s1, s2, s3, sum(100) AS INTEGER
DIM nm(100) AS STRING
INPUT n
FOR j = 1 TO n
LINE INPUT s$
c$ = MID$(s$, 1, 1)
i = 1
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s$, i, 1)
WEND
i = i + 1
c$ = MID$(s$, i, 1)
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(s, i, 1)
WEND
nm$(j) = MID$(s$, 1, i)
sum(j) = ASC(MID$(s$, i + 1, 1)) - ASC("0")
sum(j)=sum(j)+(ASC(MID$(s$,i+3,1))-ASC("0"))
sum(j)=sum(j)+(ASC(MID$(s$,i+5,1))-ASC("0"))
NEXT j
s1 = 0: s2 = 0: s3 = 0
FOR j = 1 TO n
  IF sum(j) > s1 THEN
    s3 = s2: s2 = s1
    s1 = sum(j)
  ELSE
  IF sum(j) > s2 THEN
    s3 = s2: s2 = sum(j)
  ELSE
    IF sum(j) > s3 THEN s3 = sum(j)
  END IF
  END IF
NEXT j
FOR j = 1 TO n
IF sum(j) >= s3 THEN PRINT nm$(j)
NEXT j
END
```

Указания по оцениванию	Баллы
Программа работает верно, т.е. корректно выделяет из входных данных оценки, ищет три лучших суммы баллов и распечатывает учеников, набравших эти суммы. Допускается наличие в тексте программы одной синтаксической ошибки.	4
Программа работает в целом верно, но содержит, по крайней мере, две из следующих неточностей (нерациональностей): сохраняются не суммы баллов (средние баллы), а сами баллы и суммы перевычисляются несколько раз заново; явно вычисляются средние баллы, что приводит к сравнению вещественных чисел; при нахождении трех максимальных значений элементы массива переставляются местами; при печати сравнения производятся с каждым из трех максимальных элементов. Допускается наличие от одной до трех 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.	3
Программа работает в целом верно, но выводит только трех лучших учеников, даже если кто-то еще сдал экзамены не хуже. Возможно, в реализации алгоритма содержатся 1–2 ошибки (используется знак "<" вместо ">", "or" вместо "and" и т.п.). Возможно, некорректно организовано считывание входных данных. Допускается наличие до пяти 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.	2
Программа неверно работает при некоторых входных данных и, возможно, содержит ошибку в алгоритме поиска трех максимальных элементов. Допускается до 4 различных ошибок в ходе решения задачи, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи 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.	1
Задание выполнено неверно	0
Максимальный балл	4

4.3.

Содержание верного ответа и указания по оцениванию (допускаются иные формулировки ответа, не искажающие его смысла)
Программа читает все входные данные один раз, сразу подсчитывая в массиве с индексами от 0 до 50 количество участников, набравших тот или иной балл. Путем просмотра этого массива с конца (от 50 баллов) определяется число участников, заведомо попадающих в число 45% лучших (добавление всех участников, набравших следующий балл, приводит к выходу за 45%). Последний балл, который набрали не менее 1 участника, запоминается. Если хотя бы один из следующих участников также попадает в 45%, то проверяется, что он и следующие, набравшие столько же баллов, набрали более половины баллов, в этом случае они все добавляются к числу победителей и призеров и их балл является искомым. Баллы начисляются только за программу, которая решает задачу хотя бы для одного частного случая (например, когда все участники набрали различные баллы, каждый балл кто-то набрал и 45% от числа участников – целая величина). Ниже приведены примеры решения задания на языках Бейсик и Паскаль. Допускаются решения, записанные на других языках программирования. При оценивании решений на других языках программирования необходимо учитывать особенности этих языков программирования.
Пример правильной и эффективной программы на языке Паскаль:

```
Var
  cnt: array[0..50] of integer;
  c: char;
  i, k, N, b, S, minb: integer;
begin
  for i:=0 to 50 do
      cnt[i]:=0;
  readln(N);
  for i:=1 to N do
  begin
    repeat
      read(c);
    until c=' '; {считана фамилия}
    repeat
      read(c);
    until c=' '; {считано имя}
    readln(k,b);
    cnt[b]:=cnt[b]+1;
  end;
  S:=0;
  b:=50;
  while (S + cnt[b])*100<=N*45 do
   begin
    S:=S+cnt[b];
    if cnt[b]>0 then minb:=b;
    b:=b-1
   end; {определены те, кто наверняка стал призером и пропущены баллы, которые никто не набрал}
```

```
  if (S+1)*100<=N*45 then
{если еще хотя бы один участник попадает в 45%,
  то проверяется, какой балл набрали он и следующие участники}
  begin
    if b>25 then minb:=b
  end;
  writeln(minb);
end.
```

Пример правильной и эффективной программы на языке Бейсик:

```
DIM cnt(50) AS INTEGER
FOR i = 0 TO 50
cnt(i) = 0
NEXT i
INPUT N
FOR j = 1 TO N
LINE INPUT ss$
c$ = MID$(ss$, 1, 1)
i = 1
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(ss$, i, 1)
WEND
i = i + 1
c$ = MID$(ss$, i, 1)
WHILE NOT (c$ = " ")
 i = i + 1
 c$ = MID$(ss$, i, 1)
WEND
i = i + 1
c$ = MID$(ss$, i, 1)
IF c$ = "1" THEN
  i = i + 1
ENDIF
b = VAL(MID$(ss$, i + 2))
cnt(b) = cnt(b) + 1
NEXT j
s = 0
b = 50
WHILE (s + cnt(b)) * 100 <= N * 45
  s = s + cnt(b)
  IF cnt(b)>0 THEN minb = b
  b = b - 1
WEND
IF (s + 1) * 100 <= N * 45 THEN
  IF b > 25 THEN minb = b
ENDIF
PRINT minb
END
```

Указания по оцениванию	
Программа работает верно для любых входных данных произвольного размера и находит ответ, не сохраняя все входные данные или баллы участников в массиве, размер которого равен числу участников. Программа просматривает входные данные один раз, заполняя во время считывания массив размерностью от 0 до 50 для хранения количества участников, набравших то или иное количество баллов. Искомые величины находятся путем однократного просмотра этого массива. При определении минимального балла рассматриваются только баллы, которые кто-то набрал. Допускается наличие в тексте программы одной синтаксической ошибки.	4
Программа работает верно, но все входные данные (или баллы участника) запоминаются в массиве, размер которого совпадает с количеством участников, или входные данные считываются несколько раз. Возможно, используется сортировка всех баллов участников и/или алгоритм поиска минимума, просматривающий баллы всех призеров. Допускается наличие от одной до трех 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.	3
Программа работает в целом верно, эффективно или нет, но, в реализации алгоритма содержатся 1–2 ошибки (выход за границу массива, перевод символов в числа, используется знак "<" вместо "<=", "or" вместо "and" и т.п.). Возможно, некорректно организовано считывание входных данных. Одной из двух ошибок может быть следующая логическая ошибка: при анализе результата последнего участника, вошедшего в 45%, проверяется, что и все, а не хотя бы один, набравшие вместе с ним тот же балл, попали в 45%, или в качестве пограничных участников проверяются участники, не вошедшие в 45%. Ошибкой может быть также учет баллов, которые никто не набрал. Допускается наличие от одной до пяти синтаксических ошибок, описанных выше.	2
Программа, возможно, не всегда верно определяет количество победителей и призеров и, соответственно, минимальный балл у призеров. При использовании сортировки допущены ошибки в ее реализации. Допускается до 4 различных ошибок в реализации алгоритма, в том числе описанных в критериях присвоения двух баллов. Допускается наличие от одной до семи синтаксических ошибок, описанных выше.	1
Задание не выполнено или выполнено неверно	0
Максимальный балл	*4*

СОДЕРЖАНИЕ

Отпечатано с готовых диапозитивов в филиале ГУП МО «КТ» «Воскресенская типография»
140200, г. Воскресенск Московской области, ул. Вокзальная, д. 30
E-mail: vosprint@mail.ru тел.: 8 (49644) 2-45-42 Заказ № 2825