

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

С.В. Сидоров
Г.С. Крынецкая

Методическое пособие
по подготовке к олимпиадам
школьников инженерной
направленности

**Информационно-технологическое
направление
«МАТЕМАТИКА
В ИНФОРМАТИКЕ. РЕШЕНИЕ
ОЛИМПИАДНЫХ ЗАДАЧ»**

7–8 классы



Москва 2017

УДК 681.3
К85

Крынецкая Г.С.

К85 Методическое пособие по подготовке к олимпиадам школьников инженерной направленности : Информационно-технологическое направление «Математика в информатике. Решение олимпиадных задач» : 7–8-й классы / Г.С. Крынецкая, С.В. Сидоров. – М. : Изд. Дом НИТУ «МИСиС», 2017. – 68 с.

Рассматриваются олимпиадные задачи информационно-технологического направления предыдущих лет для школьников средних классов. По каждой задаче приводится теоретическое сопровождение темы задачи, различные способы ее решения, типовые ошибки, допущенные в решении подобных задач.

Темы задач – кодирование различных видов информации, системы счисления, алгоритмизация (следование алгоритму, составление алгоритма, анализ алгоритма), программирование и математика.

В задачах программирования коды программ приводятся на языках C#, PascalABC и VBA.

Предназначено для подготовки к решению олимпиадных задач, а также для углубленного изучения информатики и математики школьниками средних классов школы.

УДК 681.3

© С.В. Сидоров,
Г.С. Крынецкая, 2017
© НИТУ «МИСиС», 2017

ОГЛАВЛЕНИЕ

Введение.....	4
Задача 1 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	5
Задача 2 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	11
Задача 3 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	12
Задача 4 (Олимпиада школьников МИСиС, 2011 год).....	14
Задача 5 (Олимпиада школьников МИСиС, 2010 год).....	15
Задача 6 (9 класс, Роботрон, отборочный тур 2016 год).....	17
Задача 7 (Роботрон, отборочный тур 2016 год).....	19
Задача 8 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 10 класс, отборочный тур).....	24
Задача 9 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	26
Задача 10 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	33
Задача 11 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс).....	43
Задача 12 (Олимпиада школьников МИСиС, 2010 год).....	54
Заключение.....	67

ВВЕДЕНИЕ

Мы являемся стремительного развития информационных технологий и их применения в самых различных отраслях человеческой деятельности. Информатика прочно вошла в нашу жизнь. Первое знакомство с информационными технологиями начинается с начальных классов школы с изучения дисциплины «Информатика» и продолжается в течение всей жизни.

Термин «информатика» включает в себя множество тем и понятий. Это и программирование сайтов, и работа с базами данных, и знание аппаратной части вычислительных систем, и сетевые технологии, и многое другое. Для успешного освоения информационных дисциплин требуется навыки алгоритмического мышления, хорошая математическая подготовка, понимание логики программ и знание типов данных.

Цель данного пособия – пробудить интерес к углубленному изучению информатики, решению неординарных задач, к программированию. Авторы надеются, что изучение данного пособия будет способствовать развитию навыков в создании и отладке кодов программ, более глубокому изучению языков программирования, умению работать с символьными и с текстовыми типами данных.

Олимпиада по информатике «МИСиС зажигает звезды» проводится для 7, 8, 9, 10 и 11 классов в два этапа: отборочный и заключительный. Варианты для всех классов различаются по уровню сложности предлагаемых задач.

В пособие приведены задания олимпиад предыдущих лет. Все задачи снабжены подробными решениями, анализом ошибок и рекомендациями по оптимизации алгоритмов. Задачи имеют разный уровень сложности, что позволяет использовать пособие для учащихся с различной степенью подготовки.

Программные реализации задач приведены на трех наиболее популярных в настоящее время языках программирования:

- Pascal ABC, как один из популярных языков, изучаемых в школах;
- C# по причине углубленного изучения в МИСиС;
- VBA (Visual Basic for Application) как наиболее распространенная версия Бейсика, входящая в состав пакета Microsoft Office.

Это позволяет заинтересованным читателям потренироваться при решении задач на всех этих языках программирования, сравнить их

возможности, почувствовать их преимущества и недостатки, убедиться, что все языки структурного программирования в целом похожи.

Пособие предназначено для школьников 7-8 классов. Основное внимание в нем уделено вопросам кодирования информации, системам счисления, задачам составления, следования и тестирования алгоритмов. Последние задачи содержат коды программных реализаций. Пособие может быть полезно студентам первых курсов и учителям информатики.

Задача 1 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Алфавит племени Юмба-Тумба невелик. В приведенном сообщении использованы все символы алфавита. Определите количество бит и количество байт, которое потребуется для кодирования сообщения. Закодируйте сообщение в двоичном коде.



Проверяется

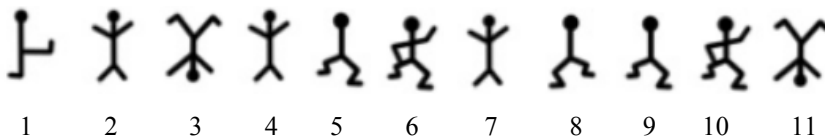
Владение темой кодирование информации. Умение кодировать информацию. Представление информации в памяти.

Обсуждается




Кодирование информации. Представление информации в памяти. Биты. Байты.


Теоретическое сопровождение темы. Решение

Рассмотрим внимательно сообщение, пронумеровав позиции символов.




Проанализировав сообщение, можно увидеть, что из 11 представленных символов, уникальными являются 6 символов:

-  – символ встречается один раз и размещается на первом месте.
-  – символ расположен на второй, четвертой и седьмой позициях.
-  – символ находится на третьей и одиннадцатой позиции.

 – занимает пятую и девятую позиции.



 – занимает шестую и десятую позиции.

 – занимает восьмую позицию.





Согласно заданию сообщение требуется перекодировать в двоичном коде. Для кодирования информации в двоичном коде надо каждому символу дать **уникальную**, соответствующую ему комбинацию нулей и единиц.

Если символы используются с одинаковой частотой, рекомендуется использовать код постоянной длины, при котором каждый символ кодируется одинаковым количеством символов двоичной системы. Определим, сколько разрядов двоичной системы требуется отвести под кодирование одного символа племени Юмба-Тумба.


Если выделить один разряд двоичной системы, то он позволит закодировать только два символа племени Юмба-Тумба. Например, так:



Кодирование в двоичной системе	Символ племени Юмба-Тумба
0	
1	

Если выделить два разряда двоичной системы, то они позволят закодировать четыре символа племени Юмба-Тумба. Два символа окажутся декодированными. Например, так:

Кодирование в двоичной системе	Символ племени Юмба-Тумба
00	
01	
10	
11	

Если выделить три разряда двоичной системы, то они позволят закодировать восемь символов племени Юмба-Тумба. Например, так:

Кодирование в двоичной системе	Символ племени Юмба-Тумба
000	

001	
010	
011	
100	
101	
110	Не используемая комбинация символов
111	Не используемая комбинация символов

Как видим, трех разрядов достаточно для кодирования шести символов. Иначе, количество разрядов n , требуемых для кодирования k символов определяется по формуле $2^n \geq k$. То есть надо определить такое n , при котором $2^n \geq k$.

Также наглядно показано, что не всегда все комбинации являются задействованными, могут остаться неиспользуемые комбинации.

Для получения сообщения, закодированного в двоичной системе надо сопоставить каждому символу племени Юмба-Тумба соответствующую комбинацию:



000 001 010 011 100 101 110 111 100 101 110

Общий объем сообщения 000001010001011100001101011100011 в битах в этом случае составит $11 \cdot 3 = 33$ бита.

Если определять объем, который займет сообщение, в байтах, то 33 бита полностью займут 4 байта и еще один бит. Но для записи последнего (33-го) бита потребуются выделить в памяти целый байт, несмотря на то, что он не будет полностью использован. Таким образом, объем сообщения в байтах – 5 байт.









Правильный ответ: 000001010001011100001101011100011; 33 бита или 5 байт на сообщение.

Естественно, соответствие символа племени Юмба-Тумба трехразрядному двоичному коду, и, как результат, закодированный вариант сообщения, может отличаться от предложенной в данном решении схемы.





Данная задача имеет альтернативный вариант решения – использование кода переменной длины.






Использование кода переменной длины оправдано для ситуаций, когда ряд символов встречается в сообщении чаще, чем прочие. В этом случае под кодирование часто используемых символов отводится малое количество разрядов двоичной системы; под кодирование редко используемых символов отводится более длинная комбинация двоичной системы. Результатом использования кода переменной длины является уменьшение объема сообщения.

При составлении таблицы соответствия с использованием кода переменной длины для обеспечения однозначного соответствия кодируемых символов очень важно соблюдать правило: комбинация символов двоичной системы, использованная для кодирования какого-либо символа племени Юмба-Тумба, не должна служить началом для другого символа племени Юмба-Тумба.

Руководствуясь этим правилом, составим таблицу соответствия символов племени Юмба-Тумба  , учитывая их распространенность в сообщении. Самым популярным является символ  , который встречается трижды. Начнем именно с этого символа. Поставим ему в соответствие короткую комбинацию «0». Кодирова остальные символы надо помнить, что никакая другая комбинация не должна начинаться с «0». Следовательно, допустимые варианты должны начинаться с «1». Менее используемому символу  (встречается 2 раза) можно поставить в соответствие «10». Если следующему символу  с поставить в соответствие «11», то на этом все допустимые комбинации оказываются исчерпанными: невозможно подобрать комбинацию, которая не будет начинаться с «0», «10» или с «11». Если же какому-то символу племени Юмба-Тумба поставить в соответствие комбинацию, начинающуюся с «0», «10» или с «11», то сложится ситуация отсутствия однозначного соответствия при расшифровке сообщения. Например,  закодируем «1110». При декодировании невозможно будет определить, что именно закодировано: символ  , (код которого «1110») или два символа  (код «11») и  ((код «10»).












Символ племени Юмба-Тумба	Количество использований	Кодирование в двоичной системе
	1	Нет комбинации
	3	0

	2	10
	2	11
	2	Нет комбинации
	1	Нет комбинации

Выберем другой путь. Выделим под символы, использующиеся два раза, три разряда:  пусть кодируется как «100». Символ  закодируем как «101».  – «110». Последняя допустимая комбинация «111», а символов племени Юмба-Тумба остаются два. Для наименее используемых символов  и  введем комбинации «1110» и «1111». Получим следующую таблицу соответствия

Символ племени Юмба-Тумба	Количество использований	Кодирование в двоичной системе
	1	1110
	3	0
	2	100
	2	101
	2	110
	1	1111







В этом случае сообщение будет кодироваться так

1110 0 100 0 101 110 0 1111 101 110 100












Общий объем сообщения 11100100010111001111101110100 в битах в этом случае составит $3 \cdot 1 + 2 \cdot 3 + 2 \cdot 3 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 = 29$

бит. Или 4 байта. Объем сообщения при введении кода переменной длины уменьшился.

Можно кодировать иным путем: самый популярный символ  кодировать двумя разрядами «00». Менее популярные  и  пусть кодируются как «01» и «10». Остается всего одна комбинация из двух символов – «11», и три незакодированных символа. В этом случае комбинация «11» не может быть использована для какого-либо символа. Для  введем – «110». Для наименее используемых символов  и  введем комбинации, которые должны начинаться с «111»: «1110» и «1111». Тогда получим

Символ племени Юмба-Тумба	Количество использований	Кодирование в двоичной системе
	1	1110
	3	00
	2	01
	2	10
	2	110
	1	1111

В этом случае сообщение будет кодироваться

1110 00 01 00 10 110 00 1111 10 110 01

Определим объем сообщения для последнего варианта: Общий объем сообщения 1110000100101100011111011001 в битах в этом случае составит $3 \cdot 2 + 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 = 28$ бит (4 байта). Последняя схема кодирования позволила еще больше сократить объем сообщения.

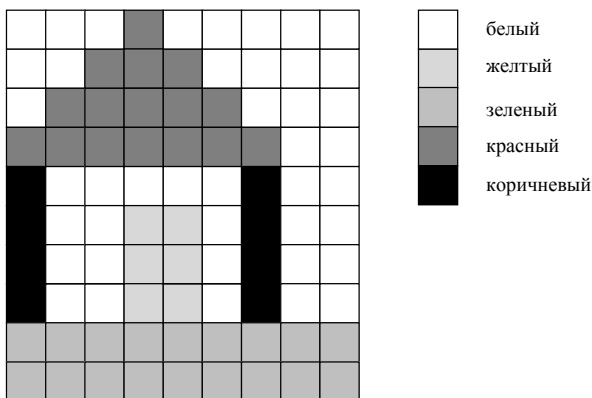
Типичные ошибки, допущенные при реализации задачи

1. Непонимание сути кодирования информации.

2. Ошибки невнимательности определения количества символов и перекодирования.
3. Некорректное использование кода переменной длины.
4. Ответ 4,125 байта. При размещении информации в памяти байт отводится целиком.

Задача 2 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Определите, какой минимальный объем потребуется для хранения закодированного изображения приведенного рисунка в двоичном коде в битах.



Проверяется

Навыки кодирования информации.

Обсуждается

Кодирование растровой графической информации.

Теоретическое сопровождение темы. Решение

В данной задаче приведено растровое изображение, в котором рисунок состоит из множества точек – пикселей. Каждая точка имеет свой цвет.

Соответственно, для кодирования такого изображения требуется описать цвет каждой точки.

Общее количество точек рисунка равно $9 \cdot 10 = 90$ (9 по ширине, 10 по высоте).

В картинке используется 5 цветов. Надо определить, какое количество разрядов двоичной системы потребуется для кодирования пяти цветов.

Для кодирования k различных объектов (точек, символов) требуется найти такую минимальную степень двойки n , при которой $2^n \geq k$.

В данной задаче для кодирования пяти символов имеем 2^3 больше пяти. То есть требуется три разряда.

Для наглядности сопоставим каждый цвет с числом:

Число	Цвет рисунка	Двоичный код числа
0	Белый	000
1	Желтый	001
2	Зеленый	010
3	Красный	011
4	Коричневый	100
6	Не используется	101
7	Не используется	110
8	Не используется	111

Отведя три разряда для записи о цвете каждой точке, получаем три неиспользуемых комбинации, что является нормальной ситуацией в подобных задачах: если отвести всего два разряда, то для одного цвета не хватит числовой комбинации.

При выбранной схеме кодирования кодированное изображение будет представляться в памяти следующей цепочкой двоичных символов:

000	000	000	011	000	000	000	000	000	000	
000	000	011	011	011	000	000	000	000	000	
000	011	011	011	011	011	000	000	000	000	
011	011	011	011	011	011	011	000	000	000	
100	000	000	000	000	000	100	000	000	000	
100	000	000	001	001	000	100	000	000	000	
100	000	000	001	001	000	100	000	000	000	
100	000	000	001	001	000	100	000	000	000	
010	010	010	010	010	010	010	010	010	010	
010	010	010	010	010	010	010	010	010	010	

Минимальный объем памяти в битах рассчитывается, как $9 \cdot 10 \cdot 3 = 270$ бит.

Типичные ошибки, допущенные при реализации задачи

Неправильное определение количество разрядов для кодирования пяти цветов как результат незнания темы кодирование графической растровой информации.

Задача 3 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Мы делим апельсины. В какой системе счисления можно разделить 10_n апельсинов между тремя (в десятичной системе счисления) друзьями поровну и без остатка и так, чтобы каждый получил только целые апельсины? Укажите **все** варианты ответов.

Проверяется

Знание систем счисления, понимание сути системы счисления.

Обсуждается

Системы счисления. Формирование математически-верных ответов.

Теоретическое сопровождение темы. Решение

Система счисления – способ записи числа. В настоящее время наиболее популярны позиционные системы счисления, в которых нахождение (позиция) символа-цифры влияет на значение числа. Например, «01» и «10» - различные по значению числа. Цифра «1» в первом случае оказывается в первом (младшем) разряде, что дает число 1. Во втором случае эта цифра во втором разряде, что увеличивает вес числа в 10 раз.

Наиболее распространенной позиционной системой счисления является десятичная система, подразумевающая использование десяти различных символов в каждом разряде («0», «1», «2», «3», «4», «5», «6», «7», «8», «9»).

Помимо десятичной системы счисления существуют и иные системы счисления. В информатике популярны двоичная, шестнадцатеричная, восьмеричная системы. Но возможны также системы и с любым иным основанием. Основание системы счисления определяет количество используемых символов в каждом разряде. Например, в пятеричной системе счисления используются символы («0», «1», «2», «3», «4»). А, если ввести двенадцатеричную систему счисления, то классических принятых цифр не хватает, потому можно использовать латинские буквы алфавита («0», «1», «2», «3», «4», «5», «6», «7», «8», «9», «A», «B»).

Обратим внимание, что наименование системы счисления указывает количество используемых символов в разряде; первым символом является ноль. Таким образом, в десятичной системе само число десять записывается как 10_{10} (задействуется следующий разряд). В двоичной системе счисления число 2 (основание системы счисления) записывается как 10_2 . Аналогичную картину можно проследить для пятеричной системы (10_5), двенадцатеричной (10_{12}) и прочих систем счисления. Значение основания системы счисления n записывается как 10_n – задействуется второй разряд.

Очевидно, что в десятичной системе счисления 10 на 3 нацело не делится. На 3 (десятичное) делятся десятичные числа 3, 6, 9, и т.д.

Самым простым вариантом решения представляется записать перечисленные числа (3, 6, 9, и т.д.) в основании системы счисления. То есть получим соответственно троичную, шестеричную, девятиричную и т.д. системы счисления.

В данной задаче важно также понимать, что перечисленными системами ответ не ограничивается; таких систем счисления бесконечное множество. Для обеспечения математической корректности ответа надо указать все варианты ответа. А так как перечислить бесконечное количество чисел невозможно, требуется указать механизм формирования каждого следующего числа.

Все вариантов ответов формируются как результат умножения целых положительных чисел на число 3 (в десятичной системе). Исходя из этого, ответ можно записать следующим образом.

Ответ: в системе счисления с основанием $3, 6, 9, 12, \dots, n$, где $n = 3 \cdot k$, k – натуральное число.

Иначе, каждое следующее основание получается как сложение предыдущего с числом 3 (в десятичной системе). Исходя из этого, ответ можно записать следующим образом.

Ответ: в системе счисления с основанием $3, 6, 9, 12, \dots, n_{k-1}, n_k$, где $n_k = n_{k-1} + 3$.

Типичные ошибки, допущенные при реализации задачи

1. Непонимание сути системы счисления.
2. Нахождение единственного варианта 3_{10} .
3. Неполный ответ как несколько перечисленных оснований системы счисления 3, 6, 9, 12.

Задача 4 (Олимпиада школьников МИСиС, 2011 год)

Какие из перечисленных чисел больше 100_{10} , какие меньше:

- a) $1000\ 0001_2$,
- b) $0110\ 000_2$,
- c) $0010\ 1001_2$,
- d) $1110\ 1001_2$,
- e) $0111\ 1111_2$,
- f) $0100\ 0011_2$?

Проверяется

Знание двоичной системы счисления, понимание веса разряда в любой системе счисления.

Обсуждается

Двоичные системы счисления, веса разрядов

Теоретическое сопровождение темы. Решение

Разряды двоичного числа соотносятся с десятичными числами следующим образом (иначе говоря, в таблице указан вес каждого разряда двоичной системы):

Номер разряда	8	7	6	5	4	3	2	1
Вес разряда	128	64	32	16	8	4	2	1

Таким образом, не производя перевод из двоичной системы счисления в десятичную можно легко увидеть, что $1000\ 0000_2$ (128_{10}) больше 100_{10} . То есть числа пунктов а) и д) получаются больше 100_{10} . В пункте е) задано число на единицу меньше, чем $1000\ 0000_2$. Следовательно, е) – также больше 100_{10} .

Значение в пункте б) $0110\ 000_2$, переводится в десятичную систему как сумма $64 + 32 = 96$. То есть значение меньше 100_{10} . Поразрядное сравнение пунктов с) и ф) с числом в пункте б) $0110\ 000_2$ показывает, что числа в них меньше, чем в б).

Следовательно, ответ

Больше 100_{10} будут числа: $1000\ 0001_2$, $1110\ 1001_2$, $0111\ 1111_2$.

Меньше 100_{10} числа: $110\ 000_2$, $0010\ 1001_2$, $0100\ 0011_2$.

Альтернативным вариантом решения может быть перевод числа 100_{10} в двоичную систему счисления. При этом рекомендуется также воспользоваться таблицей весов разрядов: $100_{10} = 64_{10} + 32_{10} + 4_{10} = 0110\ 0100_2$. Далее, произведя поразрядное сравнение, получаем тот же ответ.

Типичные ошибки, допущенные при реализации задачи

1. Математические ошибки
2. Преобразование всех чисел из двоичной системы счисления в десятичную. При этом задача может быть решена правильно, но с потерей большого количества времени.

Задача 5 (Олимпиада школьников МИСиС, 2010 год)

Определить, каким условиям должны удовлетворять исходные значения вводимых переменных, чтобы алгоритм закончил работу (не заикнулся).

Проверяется

Навыки алгоритмизации, умения работать с блок-схемами, анализировать алгоритмы, исходные данные и формализовать результаты.

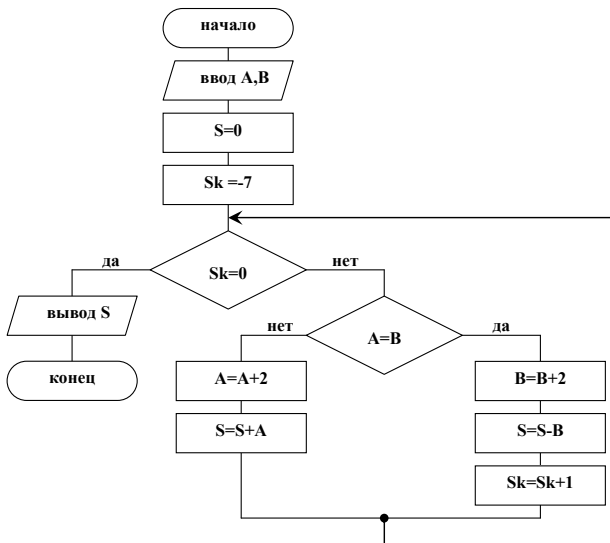
Обсуждается

Логика работы алгоритма, анализ данных

Теоретическое сопровождение темы. Решение

Анализ алгоритма показывает, что вводятся две переменные А и В, устанавливается значение счетчика $Sk = -7$. Значения переменных А и В по мере выполнения программы увеличиваются и сравниваются до

тех пор, пока счетчик не станет равным нулю. Из алгоритма видно, что счётчик увеличивается только при совпадении значений переменных А и В. Следовательно, для успешного завершения программы необходимо, чтобы значения переменных регулярно совпадали.



Если А исходно будет больше В, то выполнение программы приведет к дальнейшему увеличению А, следовательно, значения переменных уже не совпадут, счетчик не увеличится, произойдет заикливание алгоритма. Следовательно, исходное значение А не должно быть больше В.

При А равном В, алгоритм завершится благополучно. Следовательно, А может быть равно В.

При А меньше, чем В, шанс попасть на ветвь алгоритма с увеличением счетчика есть только тогда, когда значение А сможет стать равным В. Для этого разница между числами А и В должна быть четным числом: $V - A > 0$; $V - A = k$, $k = 2, 4, 6, \dots$

Получается, что либо $A = V$, либо $A < V$: $A + k = V$, k – четное целое больше нуля. Эти два условия можно объединить в одно. Окончательный ответ:

Алгоритм успешно закончит работу при $A + k = V$, k – четное целое ≥ 0

Типичные ошибки, допущенные при реализации задачи

1. Непонимание алгоритма.
2. Нахождение единственного варианта ответа $A = V$.

Задача 6 (9 класс, Роботрон, отборочный тур 2016 год)

Написать программу на любом языке высокого уровня (или нарисовать алгоритм в виде блок-схемы) для определения по двум введенным углам треугольника всех возможных следующих характеристик треугольника:

- 1) Существует ли данный треугольник
- 2) Прямоугольный, тупоугольный или остроугольный
- 3) Равнобедренный ли данный треугольник
- 4) Равносторонний ли данный треугольник

Проверяется

При решении данной задачи требуется продемонстрировать знания математики и навыки грамотного составления алгоритма для решения задачи. Важно продемонстрировать четкую логику.

Обсуждается

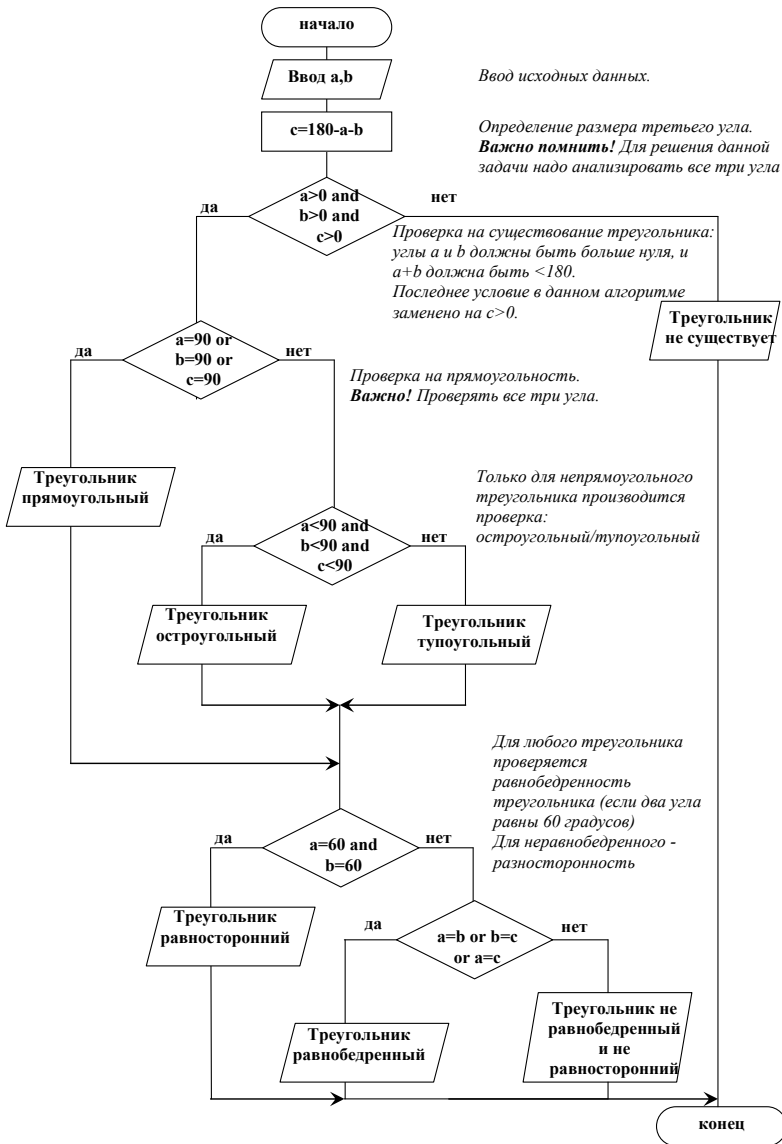
Логика работы алгоритма

Теоретическое сопровождение темы. Решение

Ниже приведен пример блок-схемы, реализующей алгоритм поставленной задачи. Естественно, ответ в данной задаче не однозначен; решения могут содержать различающиеся алгоритмы. Но в любом алгоритме обязательно

- должны проверяться все характеристики треугольника, описанные в условии;
- проверка должна производиться по трем углам; по двум заданным углам надо вычислить третий;
- должны отсутствовать математические ошибки;
- рекомендуется демонстрация четкого алгоритмического мышления;
- учитывается лаконичность алгоритма;
- учитывается грамотное построение блок-схемы. Естественно, построение алгоритма – творческий процесс, но для обеспечения понятности составленной блок-схемы важно придерживаться существующих стандартов:
 - начало и окончание алгоритма отображаются овалом;
 - ввод/вывод данных отображается параллелограммом;
 - выполнение операций отображается прямоугольником; причем в одном прямоугольнике можно перечислить несколько действий;
 - ромбом отображается ветвление алгоритма в зависимости от выполнения записанного в ромбе условия. Выходы из ромба обязательно подписываются «да/нет» (условие выполняется/ условие не выполняется)
 - линии отображают переходы от блока к блоку; при этом направление перехода (стрелку) можно не указывать при направлении слева направо и сверху вниз.

Приведенный ниже алгоритм содержит комментарии, поясняющие логику алгоритма. В приводимом решении давать комментарии не обязательно.



Приведенный алгоритм содержит некоторую избыточность: например, нет смысла проверять прямоугольный треугольник на рав-

носторонность. Но введение дополнительных ветвей алгоритма усложняет алгоритм и представляется нецелесообразным.

Типичные ошибки, допущенные при реализации задачи

1. Проверяя характеристики треугольника надо помнить, что у любого треугольника три угла и характеристики треугольника определяются этими углами. Если третий угол не задан, его необходимо вычислить

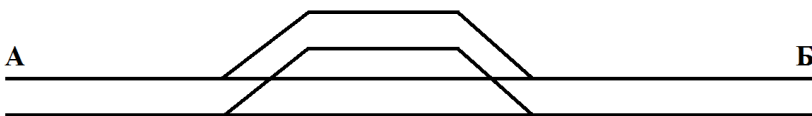
2. Логически-математическая ошибка при определении существования треугольника: вычислить третий угол по двум $c = 180 - a - b$. И далее проверять существование треугольника через достоверность равенства: $a + b + c = 180$.

3. Проверка каждого угла отдельно, что влечет значительное усложнение алгоритма и логические ошибки. Применение составных условий демонстрируют четкость мышления и умение грамотно формулировать логические условия.

4. Неграмотное, несоответствующее стандартам представление алгоритма.

Задача 7 (Роботрон, отборочный тур 2016 год)

Один поезд из восьми вагонов движется из пункта А в пункт Б, а такой же второй едет ему навстречу из пункта Б в пункт А, по тому же пути. В некоторый момент они останавливаются друг перед другом. Между поездами лежит короткий отрезок железной дороги и маленькое боковое ответвление, длина которого позволяет разместить лишь один вагон или локомотив (на рисунке схематически изображены рельсы). Каждый вагон и каждый локомотив с обеих сторон имеют устройства сцепления.



Необходимо написать алгоритм, после исполнения которого поезда смогут продолжить свой путь. Использовать команды вида: «вперед» «назад» «прицепить» «отцепить».

Проверяется

При решении данной задачи требуется продемонстрировать умение составления алгоритма, творческий склад ума и умение представить результаты.

Обсуждается

Алгоритмизация поставленной задачи

Теоретическое сопровождение темы. Решение

При решении данной задачи затруднительно дать рекомендации от чего надо отталкиваться, чтобы найти решение. Задача, как упоминалось, предполагает наличие творческого начала и умения разрабатывать нестандартные способы решения.

По духу данная задача повторяет известную задачу про две веревки и спички: каждая из веревок сгорает за 1 час, но горят они неравномерно, поэтому нельзя точно узнать, какая часть веревки за какое время сгорит. Как отмерить при помощи этих веревок интервал в 45 минут?

В упомянутой задаче про веревки и спички требуется сообразить, что веревки можно поджигать с двух концов, и требуемый результат можно получить за счет поэтапного поджигания веревок.

Возвращаясь к текущей и ей подобным задачам, можно порекомендовать использовать **все** возможности, которые заданы: если заданы команды «вперед» «назад» «прицепить» «отцепить», значит, в алгоритме надо будет расцеплять и сцеплять составы, перемещаясь вперед и назад.

Если задано, что «между поездами лежит короткий отрезок железной дороги и маленькое боковое ответвление, длина которого позволяет разместить лишь один вагон или локомотив», значит надо отцеплять и перегонять по одному вагону или локомотиву.

Если задано, что «каждый вагон и каждый локомотив с обеих сторон имеют устройства сцепления», значит, скорее всего, придется прицеплять как в начало, так и в конец состава.

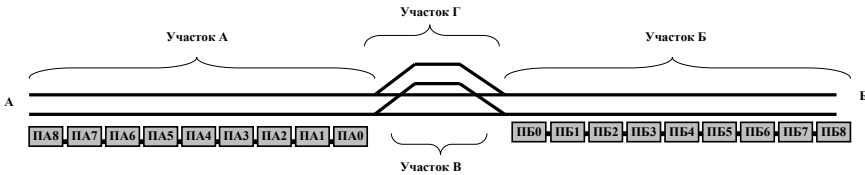
Таким образом, получив наводящие советы и подсказки из «Дано», переходим к решению задачи.

Решение:

Для удобства введем следующие обозначения:

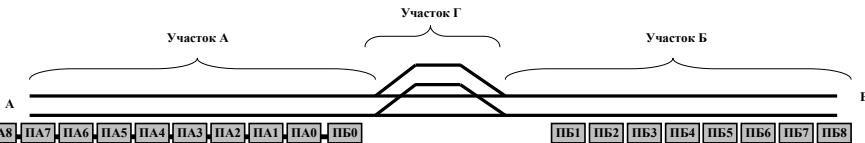
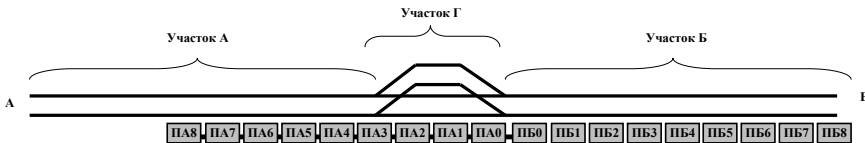
- А – участок рельсов со стороны пункта А до ответвления;
- Б – участок рельсов со стороны пункта Б до ответвления;
- В – участок основного пути между стрелками, соединяющими с ответвлением;
- Г – само ответвление;
- ПА – обозначение первого поезда, который двигался со стороны А,
 - при этом ПА0 – локомотив,
 - ПА1 – ПА8 вагоны первого поезда с первого по восьмой.
- ПБ – обозначение второго поезда, который двигался со стороны Б.
 - при этом ПБ0 – локомотив,
 - а ПБ1 – ПБ8 вагоны второго поезда с первого по восьмой.

Общая идея алгоритма следующая: вагон поезда ПБ загоняется в боковое ответвления, затем поезд ПА проезжает в Б. Прицепляет загнанный вагон к своему хвосту и перегоняет его в А. Таким образом, по одному вагону составы меняются местами.

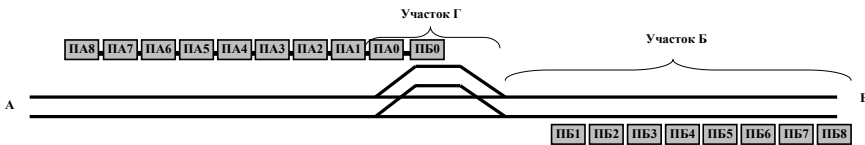


Детальное описание алгоритма:

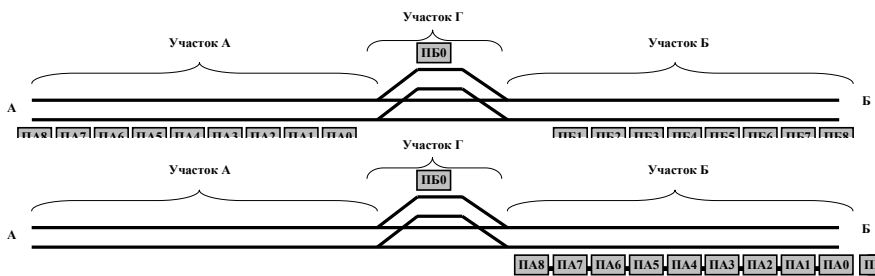
- расцепить поезд ПБ
- ПА вперед в Б
- к ПА прицепить ПБ0
- ПА назад в А



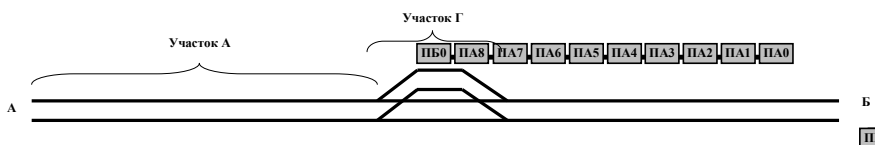
ПА вперед в Г



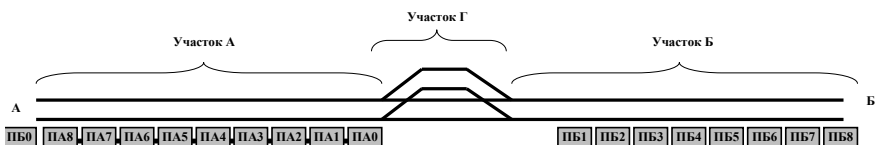
- отцепить ПБ0 в Г
- ПА назад в А
- ПА вперед в Б



ПА назад в Г
к ПА прицепить ПБ0

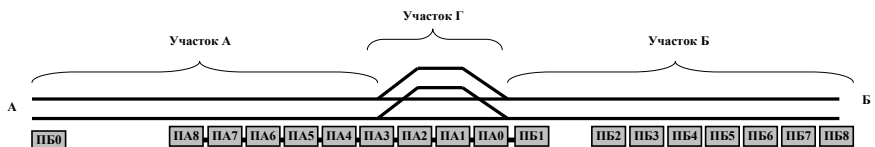


ПА назад в А
отцепить ПБк в А



По завершению перечисленных действий локомотив поезда Б окажется в нужном месте, поменявшись местами с поездом А. Легко догадаться, что далее поезд А продолжает выполнять ту же последовательность действий, чтобы по очереди перегнать все остальные вагоны:

ПА вперед в Б
к ПА прицепить ПБ1

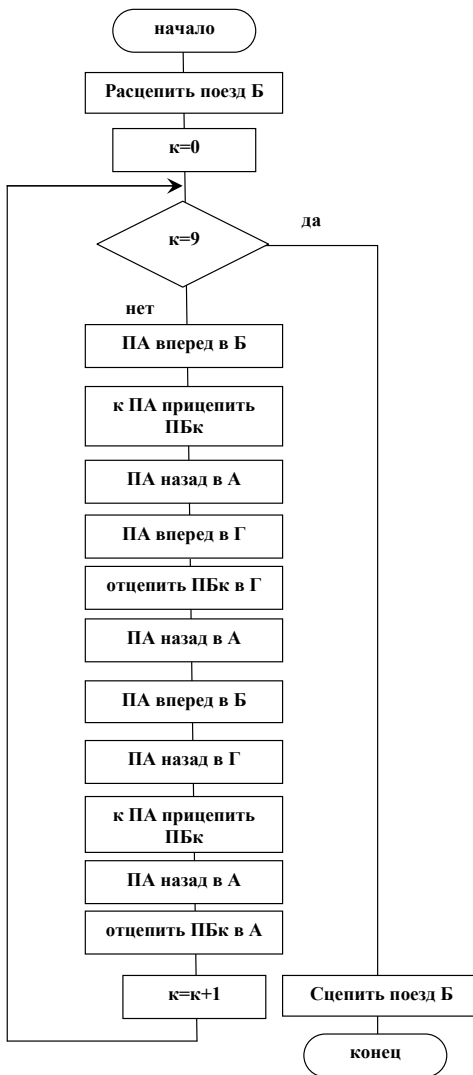


До тех пор, пока все вагоны не будут стоять слева от поезда А. Последним действием будет сцепить поезд ПБ.

Очевидно, что действия по переборке каждого вагона поезда ПБ должны выполняться в цикле. Количество повторений цикла – 9 раз (локомотив и 8 вагонов). Поэтому окончательную версию алгоритма можно записать любым из следующих способов:

1 способ: словесное описание алгоритма
 расцепить поезд ПБ
 повторить с $k = 0$ до $k = 8$
 (ПА вперед в Б
 к ПА прицепить ПБк
 ПА назад в А
 ПА вперед в Г
 отцепить ПБк в Г
 ПА назад в А
 ПА вперед в Б
 ПА назад в Г
 к ПА прицепить ПБк
 ПА назад в А
 отцепить ПБк в А)
 Сцепить поезд ПБ.

2 способ: блок-схема



Типичные ошибки, допущенные при реализации задачи

1. Пропуск тех или иных шагов алгоритма.
2. Неаккуратное оформление результатов; неиспользование цикла.

Задача 8 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 10 класс, отборочный тур)

Если вызвать некоторую функцию TEST со следующими параметрами

TEST(“МИСиС”,”С”)

TEST(“Зажигает”,”и”)

TEST(“Звезды”,”з”)

То результатом ее работы будет числа 2, 1,1.

Если вызвать некоторую функцию TEST со следующими параметрами

TEST(“Олимпиада”,”и”)

TEST(“МИСиС”,”и”)

TEST(“Звезды”,”и”)

То результатом ее работы будет числа 2, 1,0.

Какие будут результаты работы функции TEST со следующими параметрами

TEST(“Олимпийское”,”О”)

TEST(“движение”,”Д”)

Проверяется

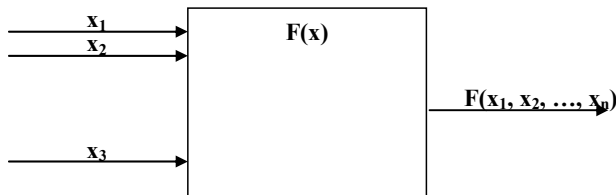
Умение выявлять закономерности, представление о функциях и параметрах функций, понимание строковых и символьных типов данных.

Обсуждается

Функции в программировании. Строковые и символьные типы данных.

Теоретическое сопровождение темы. Решение

Функция, как известно из курса математики, – это закон, согласно которому каждому набору входных параметров, ставится в соответствие единственное значение выходного параметра:



В информатике функция является самостоятельным фрагментом кода, который принимает на вход некоторое количество входных па-

раметров, а на выходе возвращает единственное значение, рассчитанное по входным параметрам.

Каждая функция возвращает значение определенного типа: в ответе может быть число, текст, логический результат и т.д.

Количество входных параметров зависит от функции; иногда параметры могут отсутствовать. Типы входных параметров (с чем именно работает функция – с числами, со строками или с иными типами данных) определяются самой функцией.

В данной задаче функция TEST, как видно из приведенных примеров, имеет два параметра: первый параметр – некая строка, второй параметр – строка длиной в один символ. На строковый тип входных параметров указывают кавычки, в которые заключаются параметры. Синтаксис большинства языков программирования и иных средств обработки данных подразумевает, что текстовая информация указывается либо в двойных кавычках, либо в одинарных.

Особенность работы с текстовой информацией заключается в том, что каждый текстовый символ кодируется соответствующим этому символу двоичным кодом в соответствии с таблицей кодирования. В качестве примеров таблиц кодирования можно назвать ASCII (каждый символ клавиатуры кодировался 1 байтом), UNICODE (для хранения одного символа отводится 2 или 4 байта).

Символ	В кодировке ASCII	В кодировке UNICODE (двоичный код)
А	1100 0000	0000 0100 0001 0000
Б	1100 0001	0000 0100 0001 0001
В	1100 0010	0000 0100 0001 0010
а	1110 0000	0000 0100 0011 0000
б	1110 0001	0000 0100 0011 0001
в	1110 0010	0000 0100 0011 0011
4	0011 0100	0000 0000 0011 0100
5	0011 0101	0000 0000 0011 0101
6	0011 0110	0000 0000 0011 0110

При составлении таблиц кодирования вводятся уникальные коды как для представления заглавных и прописных букв, так и для представления цифр, так и для прочих символов, используемых при написании текста (точка, запятая, пробел и проч.).

Для работы с символами существует специальный тип данных - символьный (char). Переменная типа char содержит всегда только один символ.

Для работы с текстом существует строковый тип данных (string). Переменная типа string содержит последовательность символов. На-

пример, строковая переменная «символ» – это последовательность символов «с», «и», «м», «в», «о», «л».

Анализируя исходные данные задачи, можно увидеть, что функция TEST всегда возвращает некоторое число, и это число совпадает с количеством символов, указанных вторым параметром, в первом параметре:

- TEST(“МИСиС”,”С”) – символ «С» встречается в слове «МИСиС» 2 раза

- TEST(“Зажигает”,”и”) – символ «и» встречается в слове «Зажигает» 1 раз

- TEST(“Звезды”,”з”) – символ «з» встречается в слове «Звезды» 1 раз. Стоит обратить внимание на то, что в данном случае буква «з» в слове «звезды» встречается дважды, но именно символ, определяющий прописную букву «з» в слове «Звезды» встречается один раз. Заглавная «З» и прописная «з» – это разные символы.

Вторая серия примеров подтверждает подмеченную закономерность первой:

- TEST(“Олимпиада”,”и”) – символ «и» встречается в слове «Олимпиада» 2 раза

- TEST(“МИСиС”,”и”) – символ «и» встречается в слове «МИСиС» 1 раз

- TEST(“Звезды”,”и”) – символ «и» в слове «Звезды» не встречается.

Следовательно, можно считать выведенную закономерность верной, и определить количество вхождений второго параметра в первый параметр:

- TEST(“Олимпийское”,”О”) – символ «О» встречается в слове «Олимпийское» 1 раз

- TEST(“движение”,”Д”) – символ «Д» в слове «движение» не встречается.

Ответ: 1,0

Типичные ошибки, допущенные при реализации задачи

1. Не увидеть закономерность.
2. Не учесть, что заглавные и прописные буквы отображаются разными символами.

Задача 9 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Робот-исполнитель снабжен камерой наблюдения и умеет шагать по плоскости; при перемещении на плоскости остается след. Робот-исполнитель имеет следующий набор команд:

Вперед (s) – по этой команде он перемещается на s шагов вперед, «куда камера глядит»

Поворот (m) – по этой команде камера поворачивается на m градусов по часовой стрелке

Повтори (k)

(

Команда 1

Команда 2

...

Команда n

) – обеспечивает повторение команд с первой по n -ую k раз.

Роботу задали следующую программу:

A = 1

Повтори (10)

(

Вперед (a)

Поворот (90)

a = a + 1

)

Поворот (90)

Повтори (10)

(

Вперед (a)

Поворот (90)

a = a – 1

)

Изобразите, что нарисует робот

Проверяется

Навыки алгоритмизации, умение следовать алгоритму, учитывать циклическое изменение параметра.

Обсуждается

Логика работы алгоритма.

Теоретическое сопровождение темы. Решение

При решении данной задачи требуется продемонстрировать умение следовать алгоритму в соответствии с предложенной системой команд вне зависимости от используемого языка программирования.

Одним из вариантов выполнения подобных задач является пошаговое выполнение всех инструкций от первой до последней команды. Но подобный подход осложнен тем, что при большом количестве команд (более семи) человеческий мозг, в отличие от компьютера,

может упустить какую-либо команду или значение какого-либо параметра, что приведет к неверному ответу.

Альтернативным вариантом решения является изучение структуры предложенного алгоритма для выявления закономерностей (циклов) и дальнейшего использования выявленных закономерностей для получения ответа.

Оценим структуру задачи:

$$a = 1$$

Повтори (10)

(

Вперед (a)

Поворот (90)

$$a = a + 1$$

)

Поворот (90)

Повтори (10)

(

Вперед (a)


Поворот (90)

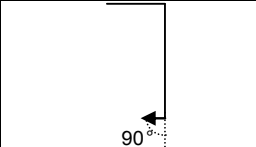
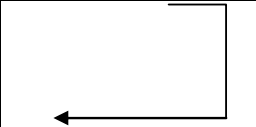

$$a = a - 1$$

)




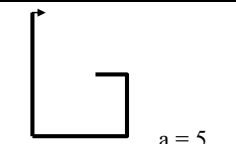
Как видно, в задаче используется два цикла. Эти циклы сходны между собой; отличие заключается в команде $a = a + 1$ для первого цикла и команде $a = a - 1$ для второго цикла. То есть в первом цикле a увеличивается на единицу, а во втором – уменьшается.



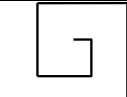

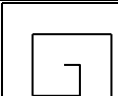

Рассмотрим, что выполняет каждый из циклов, учитывая, что значение параметра a до начала цикла равно 1 (стрелка на рисунке указывает направление камеры):

Фрагмент кода	Результат выполнения	Описание результата
Вперед (a)		Продвижение робота вперед на 1 шаг
Поворот (90)		Поворот камеры на 90 градусов по часовой стрелке
$a = a + 1$	$a = 1 + 1 = 2$	Вычисление и присвоение нового значения переменной a
Вперед (a)		Продвижение робота вперед на a шагов ($a = 2$)

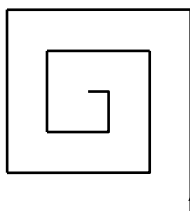
Поворот (90)		Поворот камеры на 90 градусов по часовой стрелке
$a = a + 1$	$a = 2 + 1 = 3$	Вычисление и присвоение нового значения переменной a
Вперед (a)		Продвижение робота вперед на a шагов ($a = 3$)
Поворот (90)		Поворот камеры на 90 градусов по часовой стрелке
$a = a + 1$	$a = 3 + 1 = 4$	Вычисление и присвоение нового значения переменной a

Рассмотренные три повтора цикла из десяти показывают, что на каждом шаге цикла робот идет вперед на число шагов, заданное заранее, поворачивает камеру на 90 градусов по часовой стрелке, после чего увеличивает количество шагов для последующего входа в цикл. В результате работы цикла рисуется «квадратная спираль» из десяти сторон; размер первой стороны – 1 шаг; последней – 10 шагов. По окончании цикла значение параметра $a = 11$; камера смотрит направо относительно последней нарисованной стороны:

Шаги первого цикла	Результат выполнения (на каждом новом шаге рисунок приводится в масштабе)
1 шаг	
2 шаг	
3 шаг	
4 шаг	

5 шаг		$a = 6$
6 шаг		$a = 7$
7 шаг		$a = 8$
8 шаг		$a = 9$
9 шаг		$a = 10$
10 шаг		$a = 11$

Важно не пропустить, что после окончания первого цикла расположена команда поворота камеры на 90 градусов, после которой камера будет смотреть в направлении обратном рисованию последней стороны:

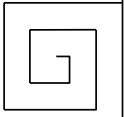
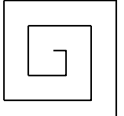
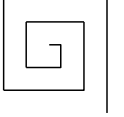
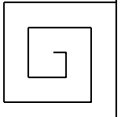
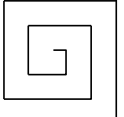
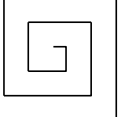


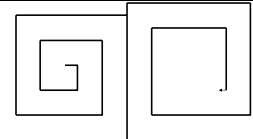
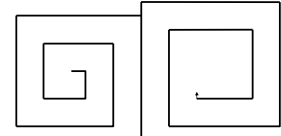
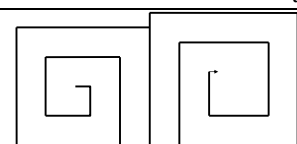
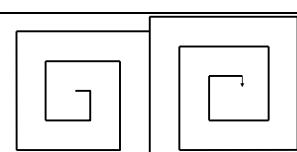
Следующий цикл сходен с предыдущим с тем отличием, что после рисования очередной стороны, длина стороны для следующего входа в цикл уменьшается. Легко понять, что в результате выполнения второго цикла будет прорисовываться «закручивающаяся квадратная спираль». Первая сторона ее наложится на уже нарисованную

последнюю сторону первой спирали с той разницей, что длина последней стороны первой спирали была равна 10 шагов, а первой стороны второй спирали – 11 шагов.

Второй цикл также выполняется 10 раз, поэтому самая последняя сторона будет равна, после чего значение параметра a примет значение 1, что на рисунке уже не отразится.

Рассмотрим детально выполнение второго цикла.

Шаги второго цикла	Результат выполнения (на каждом новом шаге рисунок приводится в масштабе)
1 шаг	 <p style="text-align: right;">$a = 10$</p>
2 шаг	 <p style="text-align: right;">$a = 9$</p>
3 шаг	 <p style="text-align: right;">$a = 8$</p>
4 шаг	 <p style="text-align: right;">$a = 7$</p>
5 шаг	 <p style="text-align: right;">$a = 6$</p>
6 шаг	 <p style="text-align: right;">$a = 5$</p>

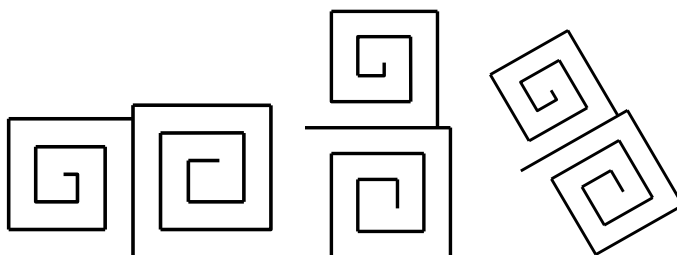
7 шаг		$a = 4$
8 шаг		$a = 3$
9 шаг		$a = 2$
10 шаг		$a = 1$

Последняя нарисованная фигура является решением данной задачи. Детальное пошаговое выполнение всех шагов программы приведено для наглядности решения.

Типичные ошибки, допущенные при реализации задачи

1. Ответ о невозможности определения результата работы робота из-за отсутствия информации об исходном направлении камеры. В задании действительно не указано исходное направление камеры, но исходное направление камеры влияет только на угол размещения фигуры конечного результата, но не оказывает никакого влияния на рисуемую фигуру.

Например, непринципиально, какое изображение из нижеприведенных будет дано в качестве ответа:



2. Непонимание алгоритма задачи.

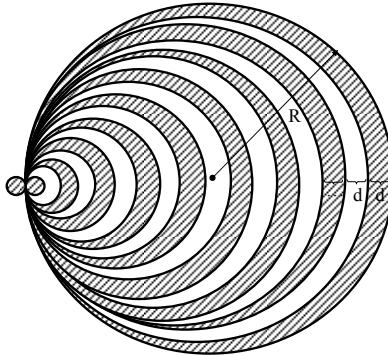
3. Ошибки невнимательности, возникающие при ручном пошаговом выполнении алгоритма, которых можно избежать, проведя предварительный анализ кода.

Задача 10 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Написать программу (алгоритм действий, блок-схему) определения площади штрихованной фигуры, если радиус самой большой окружности равен R , каждая следующая окружность имеет радиус на d меньше предыдущей, а радиусы самых маленьких окружностей совпадают?

Проверяется

Навыки алгоритмического мышления, умения видеть закономерности, использовать подмеченные закономерности для упрощения решения задачи, составлять блок-схемы и программировать, математическая подготовка, внимательность.



Обсуждается

Использование цикла для решения задач.

Теоретическое сопровождение темы. Решение

Предложенная фигура состоит из кругов; площадь круга определяется по формуле $S = \pi R^2$. В задаче задан радиус только первой (самой большой) окружности, радиус следующей окружности требуется вычислить. Видно, что задана разница между диаметрами двух соседних окружностей – d ; следовательно, радиус меньшей окружности $R_2 = R_1 - d/2$. Радиус следующей окружности можно определить как $R_3 = R_2 - d/2$ или как $R_3 = R_1 - 2 \cdot d/2$. Аналогично можно вычислить радиусы всех окружностей.

Внимательно рассмотрев предложенную фигуру можно увидеть, что площадь этой фигуры можно получить, если из площади первого

круга (πR^2) вычесть площадь второго (меньшего диаметра) круга ($\pi(R - d/2)^2$), прибавить площадь третьего круга ($\pi(R - 2 \cdot d/2)^2$), вычесть площадь четвертого ($\pi(R - 3 \cdot d/2)^2$), и т.д., ... , прибавить удвоенную площадь семнадцатого круга ($2 \cdot \pi(R - 16 \cdot d/2)^2$).

То есть площадь заштрихованной фигуры определяется по формуле:

$$S_{\text{фигуры}} = \pi R^2 - \pi(R - d/2)^2 + \pi(R - 2 \cdot d/2)^2 - \pi(R - 3 \cdot d/2)^2 + \dots + 2 \cdot \pi(R - 16 \cdot d/2)^2.$$

Требуется разработать алгоритм расчёта этой формулы.

По сути формулы (площадь формируется сложением ряда величин) требуется выполнить классический алгоритм определения суммы.

После ввода исходной информации (заданный радиус большой окружности, разница между диаметрами соседних окружностей) и определения констант (число π) сумма обнуляется и устанавливается количество повторений цикла.

В цикле вычисляется новое слагаемое (площадь следующей фигуры), условно обозначенная как $s!$, вычисленное слагаемое добавляется к сумме; значение счетчика повторений цикла уменьшается на единицу.

Цикл повторяется до тех пор, пока счетчик не станет равным нулю.

Площадь маленькой окружности повторяется, следовательно, последнее слагаемое (уже вычисленное в последнем входе в цикл) надо добавить к сумме повторно. После чего накопленная сумма выводится на печать.

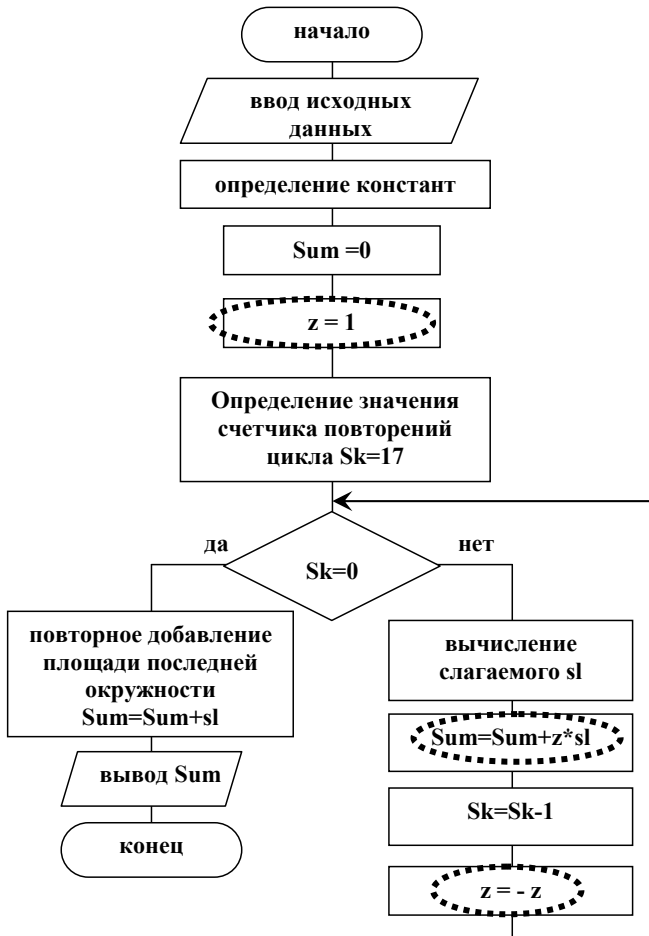
Условно алгоритм определения суммы можно представить следующим образом:



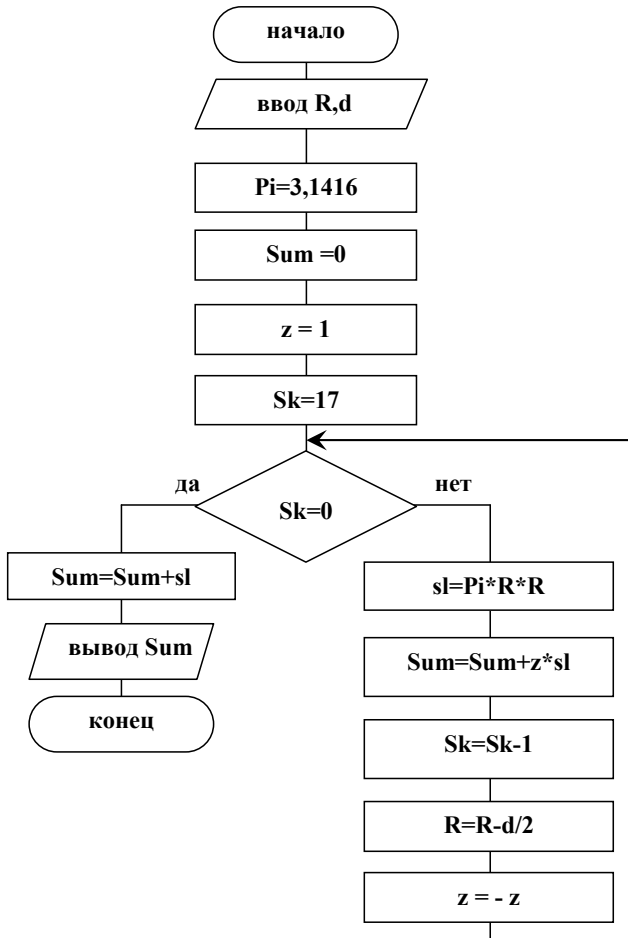
В предложенном алгоритме есть одна сложность: ряд слагаемых знакопеременный. То есть первое слагаемое положительно, второе отрицательно, третье – положительно, четвертое – отрицательно и т.д.

Решить эту проблему можно двумя путями.

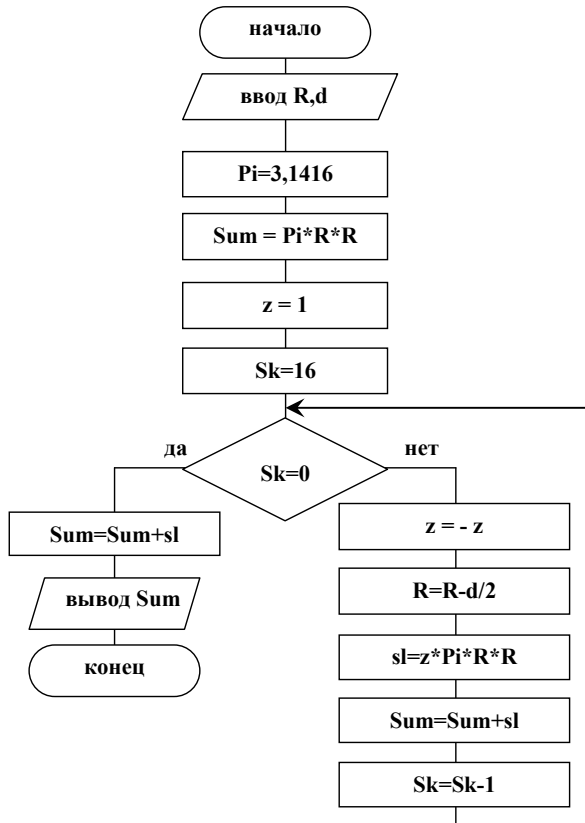
В первом случае можно ввести дополнительную переменную z , которая будет определять знак слагаемого. Исходное значение $z = 1$. При выходе из цикла значение z надо менять на противоположное $z = -z$. Каждое слагаемое в цикле должно умножаться на знак z .



При вычислении слагаемого используется одна и та же формула. А радиус для каждой следующей окружности должен уменьшаться. Таким образом, блок-схема решения данной задачи (являющаяся частью ответа) может выглядеть таким образом:



Можно реализовать данный алгоритм, уменьшив количество входов в цикл и несколько изменив порядок вычислений: сумма исходно равна первому слагаемому, количество входов в цикл – 16, знак и радиус меняют свои значения до вычисления нового слагаемого:

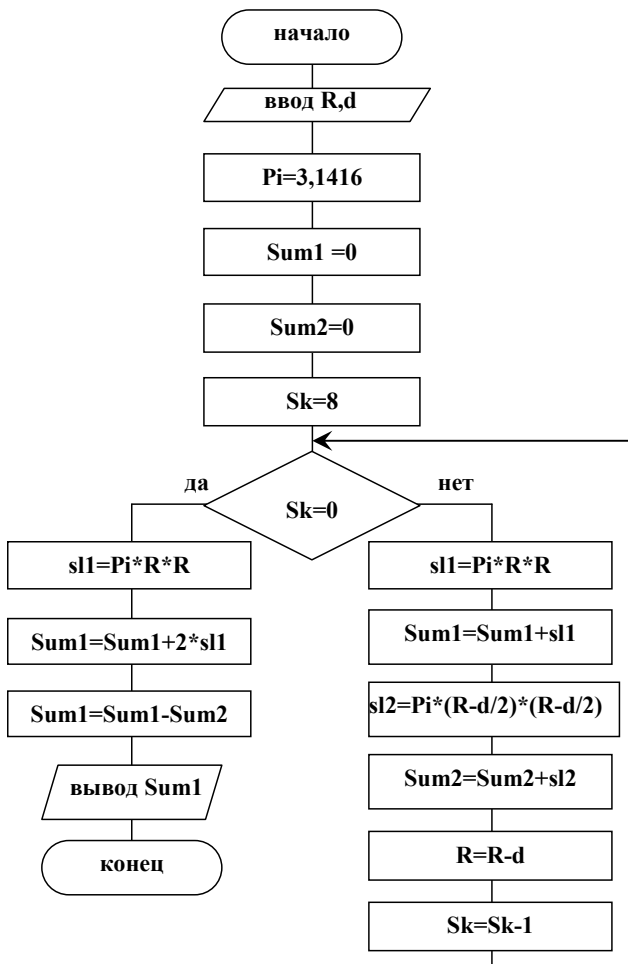


Альтернативным вариантом решения проблемы знакопеременного ряда слагаемых может быть накопление двух сумм: положительных слагаемых и отрицательных слагаемых.

Последний вариант представляется более громоздким по количеству переменных и выполняемым расчетам, но оба варианта являются правильным ответом. Естественно, в качестве ответа можно привести и любой рабочий вариант блок-схемы алгоритма.

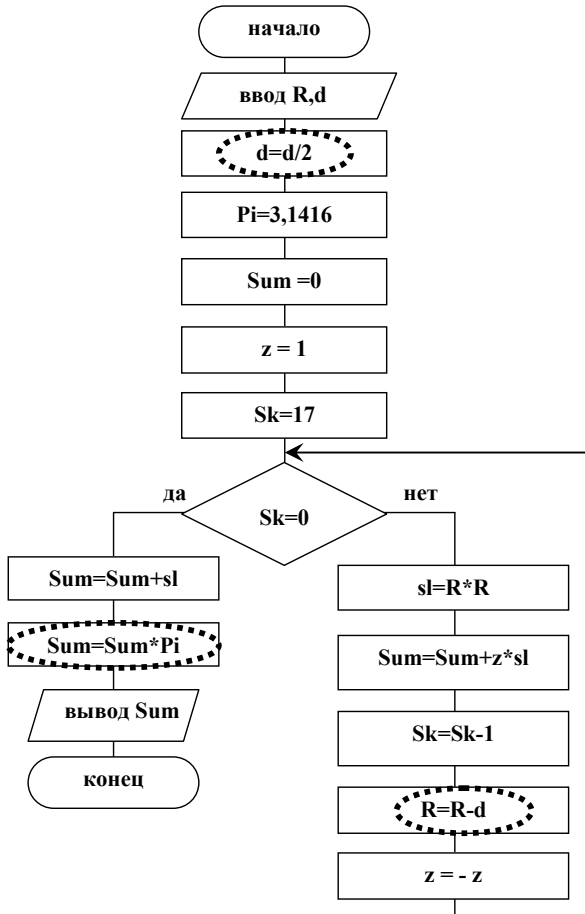
Если анализировать алгоритм с позиции времени выполнения алгоритма процессором, то можно заметить, что каждое слагаемое умножается на число π в цикле. Если вынести множитель π за скобки, то операция умножения на π будет выполнена один раз, а не 17.

Аналогично, во избежание семнадцатикратного деления параметра d пополам, можно рассчитать разницу между радиусами однократно, до цикла.



Хотя в блок-схеме появляются дополнительные шаги умножения суммы на π и деления d пополам, с точки зрения экономии времени процессора данный вариант предпочтительнее.

В данной задаче предлагается также продемонстрировать навыки программирования. Отсутствие программного кода в ответе уменьшает количество баллов за решение задачи. Далее приводится программная реализация данной задачи.



PascalABC

```

var r,d,s,sl:double;
z,i:integer;
begin
writeln('Введите радиус');
readln(r);
writeln('Введите разницу между диаметрами');
readln(d);
d:=d/2;
s:= 0;
z:= 1;
for i:= 1 to 17 do
begin
sl:= r * r;

```



```

s:= s + z * sl;
z:= -z;
r:= r - d;
end;
s:= s + sl;
s:= s * 3.1416;
writeln(s);
end.
VBA
Sub square ()
Dim r As Double
Dim d As Double
Dim s As Double
Dim sl As Double
Dim z As Integer
Dim i As Integer
r = InputBox("Введите радиус")
d = InputBox("Введите разницу между диаметрами") / 2
s = 0
z = 1
For i = 1 To 17 Step 1
    sl = r * r
    s = s + z * sl
    z = -z
    r = r - d
Next i
s = s + sl
s = s * 3.1416
MsgBox (s)
End Sub

```

C#

```

static void Main(string[] args)
{
    Console.WriteLine("Введите радиус");
    double r = double.Parse(Console.ReadLine());
    Console.WriteLine("Введите разницу между диаметрами");
    double d = double.Parse(Console.ReadLine())/2;
    double sl=0;
    double s = 0;
    int z = 1;
    for (int i = 0; i < 17; i++)
    {
        sl = r * r;
        s = s + z * sl;
        z = -z;
        r = r - d;
    }
    s = s + sl;
}

```

```

s = s * Math.PI;
Console.WriteLine("Площадь фигуры равна {0}",s);
}

```

Можно увидеть некоторые отличия в программной реализации задачи в зависимости от языка программирования. В PascalABC переменные описываются до программного кода; в C# описание переменных допускается непосредственно в программном коде; в VBA описание переменных не обязательно (программа будет работать при отсутствии операторов описания переменных dim).

Стоит обратить внимание, что, несмотря на необязательность описания переменных, грамотное составление программы подразумевает наличие описания переменных.

Типичные ошибки, допущенные при реализации задачи

1. Неиспользование цикличности в ответе. С математической точки зрения ответ может быть сформулирован как полная формула

$$\begin{aligned}
S_{\text{фигуры}} = & \pi R^2 - \pi(R - d/2)^2 + \pi(R - 2 \cdot d/2)^2 - \pi(R - 3 \cdot d/2)^2 + \\
& + \pi(R - 4 \cdot d/2)^2 - \pi(R - 5 \cdot d/2)^2 + \pi(R - 6 \cdot d/2)^2 - \pi(R - 7 \cdot d/2)^2 + \\
& + \pi(R - 8 \cdot d/2)^2 - \pi(R - 9 \cdot d/2)^2 + \pi(R - 10 \cdot d/2)^2 - \pi(R - 11 \cdot d/2)^2 + \\
& + \pi(R - 12 \cdot d/2)^2 - \pi(R - 13 \cdot d/2)^2 + \pi(R - 14 \cdot d/2)^2 - \\
& - \pi(R - 15 \cdot d/2)^2 + 2 \cdot \pi(R - 16 \cdot d/2)^2
\end{aligned}$$

С точки зрения информатики данный ответ некорректен:

- слишком громоздкая формула, при наборе которой в программе можно допустить сложно отслеживаемые ошибки;
- избыточно-большое количество математических операций, которые замедляют выполнение программы; их можно сократить, например, за счет вынесения множителя π за скобки;
- подобное линейное решение не подразумевает алгоритмизации задачи, следовательно, не может быть оценено на большое количество баллов;
- при написании программы программисту рекомендуется стремиться к созданию универсального кода, который легко может быть изменен при изменении входных данных; именно использование циклов позволяет решить задачу в общем виде при сколь угодно большом увеличении количества окружностей.

2. Математические ошибки, в частности радиус меньшего круга определяется из радиуса большего круга как $R_n = R_{n-1} - d/2$, а не $R_n = R_{n-1} - d$. Параметр d показывает, насколько диаметр меньшего круга меньше диаметра большего.

3. Ошибки невнимательности, в частности потеря в ответе удвоения площади самой маленькой окружности.

4. Небрежное составление блок-схемы кода программы, в частности отсутствие в алгоритмах и в блок-схемах ввода параметров R, d и вывода результата.

Задача 11 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 9 класс)

Встретились четыре друга футболиста и стали обсуждать количество забитых голов. Долго считали, и получили интересный итог:

- Все игроки забили разное количество голов, но количество забитых голов каждого делится без остатка и на 3 и на 2.
- Никто не забил больше ста голов.
- Разница между голами второго и первого футболиста больше разницы между голами третьего и второго футболиста.
- Разница между голами третьего и второго футболиста больше, чем разница между голами четвертого и третьего футболиста.
- Сумма голов второго и первого футболиста больше голов третьего футболиста и меньше голов четвертого футболиста.
- Напишите оптимальную по времени выполнения программу (алгоритм, блок схему) для определения, вариантов количества голов забитых каждым игроком.

Проверяется

Навыки алгоритмизации, составления блок-схем, программирования, уровень математической подготовки, умение анализировать исходные данные и результаты, понимание оптимальности выполнения программы.

Обсуждается

Математический анализ задачи. Программирование задачи. Реализация оптимального по времени выполнения алгоритма.

Теоретическое сопровождение темы. Решение

Данная задача является одной из последних (и наиболее трудоемких) в финальном туре. При выставлении баллов за решение данной задаче учитывается как математический анализ задания, так и его программная реализация. При частичной реализации задания есть шанс получить некоторое количество баллов.

Рассмотрим решение данной задачи.

Решением задачи должен стать программный код (или алгоритм), определяющий, сколько имеется «вариантов количества голов забитых каждым игроком». Следовательно, косвенно требуется рассчитать, сколько голов забил каждый из игроков.

Если в задаче сказано, что «встретились четыре друга футболиста», следовательно, имеет смысл ввести четыре переменных, значе-

ние каждой из которых будет соответствовать количеству забитых голов каждого футболиста.

Пусть g_1 – количество голов, забитых первым футболистом, g_2 – вторым, g_3 – третьим, g_4 – четвертым.

Имеется ряд условий, которые ограничивают значения переменных g_1, g_2, g_3, g_4 . Рассмотрим эти условия и формализуем их :

- «Все игроки забили разное количество голов, но количество забитых голов каждого делится без остатка и на 3 и на 2». Следовательно

$$g_1 \neq g_2 \neq g_3 \neq g_4$$

любая переменная (g_1, g_2, g_3, g_4) при делении на 2 и при делении на 3 дает в результате целое число.

- «Никто не забил больше ста голов». Следовательно

$$g_1 < 101,$$

$$g_2 < 101,$$

$$g_3 < 101,$$

$$g_4 < 101.$$

- «Разница между голами второго и первого футболиста больше разницы между голами третьего и второго футболиста». Следовательно, можно записать:

$$g_2 - g_1 > g_3 - g_2$$

- «Разница между голами третьего и второго футболиста больше, чем разница между голами четвертого и третьего футболиста». Следовательно, можно записать:

$$g_3 - g_2 > g_4 - g_3$$

- «Сумма голов второго и первого футболиста больше голов третьего футболиста и меньше голов четвертого футболиста». Следовательно, можно записать:

$$g_2 + g_1 > g_3$$

$$g_2 + g_1 < g_4$$

Количество забитых голов не может быть отрицательным числом, потому из последнего рассмотренного условия получается, что четвертый футболист забил наибольшее количество голов:

- больше, чем первый и второй вместе, следовательно, больше и первого и второго;

- больше третьего, так как сумма первого и второго больше, чем забил третий.

Если «разница между голами третьего и второго футболиста больше, чем разница между голами четвертого и третьего футболиста», следовательно, третий футболист отстал на количество забитых голов от четвертого не столь существенно, как второй отстал от третьего.

Если «разница между голами второго и первого футболиста больше разницы между голами третьего и второго футболиста», следовательно, второй футболист отстал на количество забитых голов от третьего не столь существенно, как первый отстал от второго.

Из всего изложенного, учитывая, что «все игроки забили разное количество голов», имеем:

$$g1 < g2 < g3 < g4.$$

Формализовав условия задачи, можно наметить алгоритм ее решения.

Дальнейшее аналитическое решение задачи упирается в невозможность нахождения решения множества неравенств при отсутствии уравнений. Попытка угадать возможный набор переменных займет слишком много времени. Используя программирование данную задачу можно решить прямым перебором всех значений $g1$, $g2$, $g3$ и $g4$, и поиску допустимого набора переменных, при котором соблюдаются все поставленные условия.

Подобных наборов переменных может быть несколько. Для подсчета их количества вводится специальная переменная (назовем sk), которая служит счетчиком количества найденных наборов переменных. Начальное ее значение равно нулю; при нахождении набора переменных, удовлетворяющих всем условиям, ее значение увеличивается на единицу.

Беглый анализ приведенной блок-схемы подсказывает, что разработан не самый удачный вариант алгоритма:

1. При подобном прямом переборе будут формироваться наборы значений переменных заведомо не удовлетворяющих условию; например набор 90 60 30 6 не имеет смысла даже рассматривать, так как в нем не соблюдается $g1 < g2 < g3 < g4$. На формирование данного набора переменных будут непроизводительно тратиться ресурсы вычислительной машины.

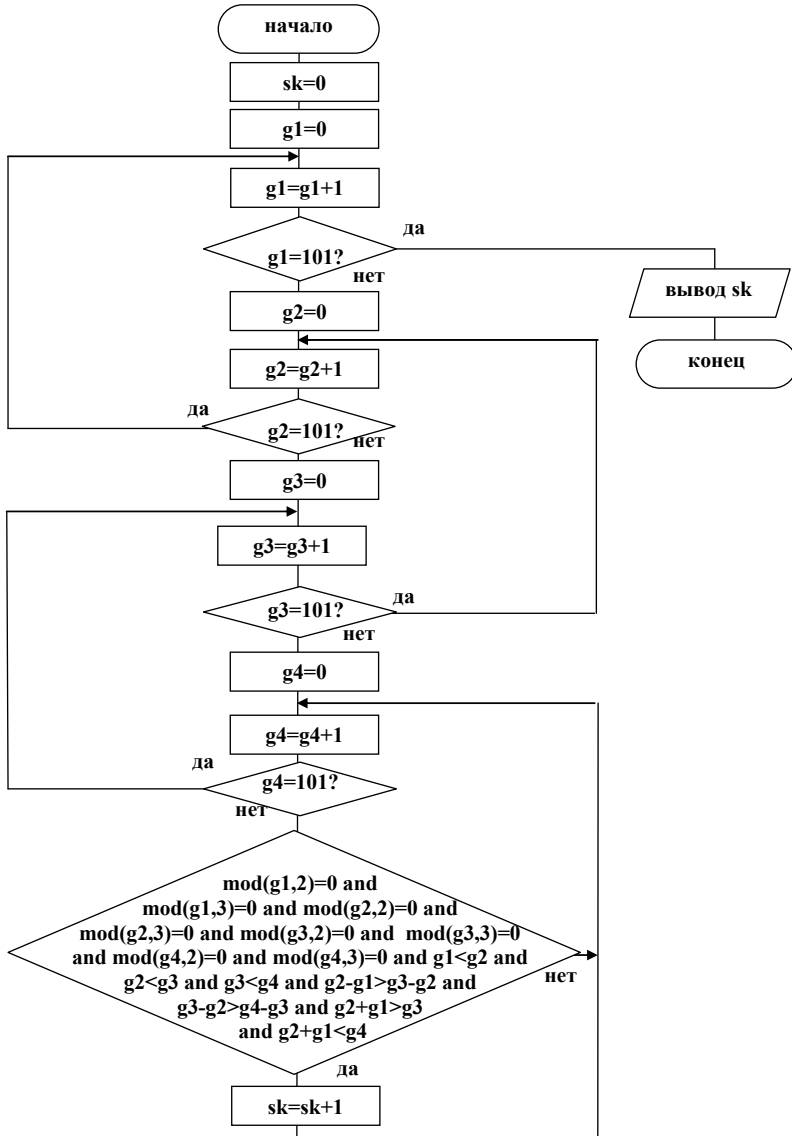
2. Из анализа чисел, очевидно, что если количество голов «делится без остатка и на 3 и на 2», то нет смысла рассматривать наборы 1 5 7 13, числа в которых не кратны ни 2, ни 3.

3. Слишком громоздкое и ресурсоемкое основное проверяемое условие.

Данная блок-схема приведена как классический пример неудачного решения: громоздкого и неоптимального. Для исправления алгоритма решения задачи можно ввести в блок-схему следующие исправления:

1. Формировать исходные наборы переменных так, чтобы исходное значение следующей переменной формировалось исходя из значения предыдущей переменной.

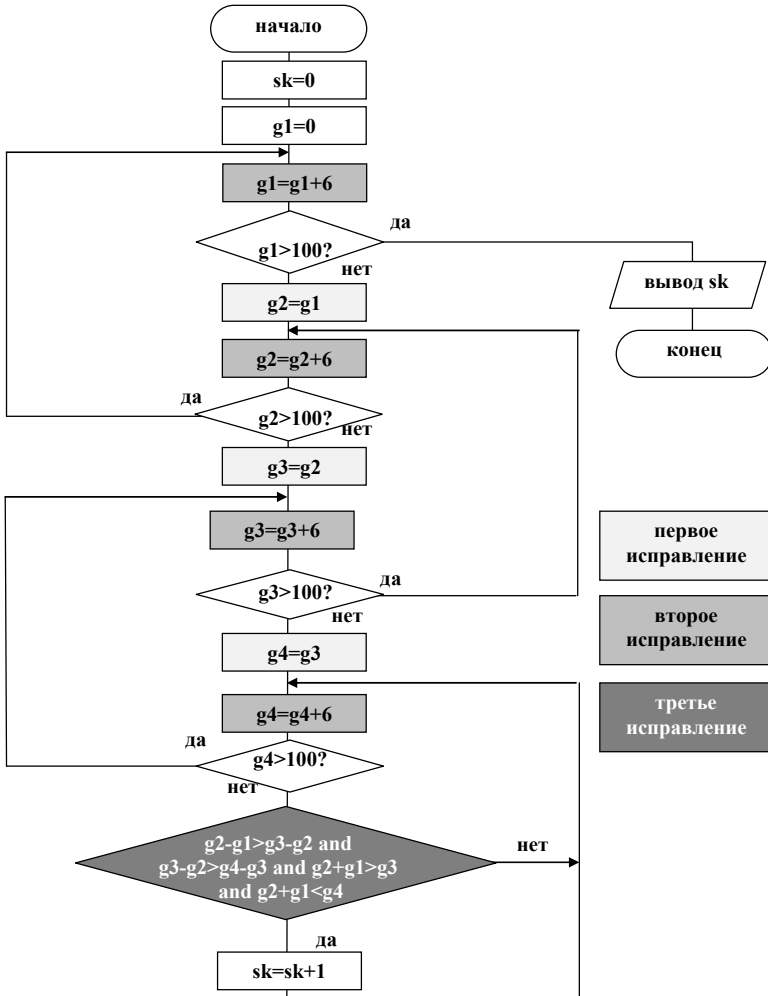
Блок-схема такого алгоритма может иметь следующий вид:



2. Чтобы каждое значение переменной было кратно как 2, так и 3, каждое следующее значение переменной получать из предыдущего, добавляя шаг 6.

3. Выполнив предыдущий шаг исправлений основное проверяемое условие можно значительно упростить, убрав проверку на кратность 2 и 3, а также – убрав проверку на $g1 < g2 < g3 < g4$.

Таким образом, блок-схема будет иметь следующий вид (исправленные блоки выделены цветом):



Несмотря на громоздкий вид блок-схемы, ее программная реализация относительно несложна: используется трехкратное вложение циклов по счетчику и проверка условия.

Приведем программную реализацию последней блок-схемы. Обратим внимание на то, что в программной реализации вводятся переменная `step`, содержащая значение, на которое изменяется количество забитых голов, и переменная `max`, которая содержит максимально-возможное число забитых голов.

Подобный подход обеспечивает гибкость и простоту модификации программы. Например, если возникает необходимость проверить работу программы при максимально-возможном количестве голов равном 200, то достаточно внести одно исправление: поменять значение переменной `max`. При отсутствии подобной переменной пришлось бы производить 4 исправления в программе.

```
C#:  
static void Main(string[] args)  
{  
    int g1, g2, g3, g4; //описание переменных g1, g2, g3, g4  
                        //как целочисленных значений  
    int step = 6;      //описание переменной step и  
                        //присваивание ей значения 6  
    int max = 100;    //описание переменной max и  
                        //присваивание ей значения 100  
    int sk = 0; ;     //описание переменной sk и  
                        //присваивание ей значения 0  
    for (g1 = 0; g1 < max;g1= g1 + step)  
//изменение значение g1 от нуля до max с шагом step  
        for (g2 =g1+step; g2 < max; g2 = g2 + step)  
//изменение значение g2 от g1+step до max с шагом step  
            for (g3 = g2+step; g3 < max; g3 = g3 + step)  
//изменение значение g3 от g2+step до max с шагом step  
                for (g4 = g3+step; g4 < max; g4 = g4 + step)  
//изменение значение g4 от g3+step до max с шагом step  
                    if ((g2 - g1 > g3 - g2) && (g3 - g2 > g4 - g3) && (g1 + g2) >  
g3 && (g1 + g2) < g4) //проверка соблюдения условий  
                        sk++; //при соблюдении условий счетчик  
                            //увеличивается на единицу  
                    Console.WriteLine("всего {0} вариантов",sk); //печать результата  
                }  
            }  
        }  
    }  
}
```

PascalABC имеет особенность реализации цикла `for`: постоянный шаг изменения переменной счетчика, равный единице. Учитывая данную особенность, будем считать переменные `g1`, `g2`, `g3` и `g4` не количеством голов, а шестой частью количества забитых голов. Переменные `step` и `max` также получают значения в шесть раз меньше. Меняем переменные `g1`, `g2`, `g3` и `g4` от их начального значения до $100/6$, но при этом в операторе условия `if` оперируем реальными значениями забитых голов, как значения `g1`, `g2`, `g3` и `g4`, умноженные на 6:


```

var g1,g2,g3,g4,sk,step,max:integer;
begin
step:= 1;
sk:= 0;
max:=100 div 6;
for g1:= 0 to max do
  for g2:= g1+step to max do
    for g3:= g2+step to max do
      for g4:= g3+step to max do
        if (g2*6 - g1*6 > g3*6 - g2*6) and (g3*6 - g2*6 >
g4*6 - g3*6) and (g1*6 + g2*6 > g3*6) and (g1*6 + g2*6 <
g4*6) then sk:=sk+1;
        writeln(sk);
      end.
    end.
  end.
end.

```

Приведенный код на PascalABC выполнен исходя из соображений максимального соответствия коду на C#, а также – блок-схеме. Но, если внимательно рассмотреть условие оператора if, то видно, что каждое неравенство не изменится, будучи сокращенным на 6. Более того, эти неравенства надо сократить; использование дополнительной операции умножения (введенной для наглядности сравниваемых величин) увеличивает время выполнения команды сравнения. Таким образом, код может быть переписан следующим образом:

```

var g1,g2,g3,g4,sk,step,max:integer; //описание всех пе-
ременных
begin
step:= 1; //переменная step определяется как 6/6=1
sk:= 0; //переменная sk исходно равна 0
max:=100 div 6; //переменная max равна целой части от деления
//100 на 6
for g1:= 0 to max do
  for g2:= g1+step to max do
    for g3:= g2+step to max do
      for g4:= g3+step to max do
        if (g2 - g1 > g3 - g2) and (g3 - g2 > g4 - g3) and
(g1 + g2 > g3) and (g1 + g2 < g4) then sk:=sk+1;
        writeln(sk);
      end.
    end.
  end.
end.

```

При этом надо помнить, что переменные g1, g2, g3 и g4 содержат значения, соответствующие шестой части забитых голов.

Чтобы написать в PascalABC программу с реальными значениями всех переменных, можно использовать цикл по условию while:

```

var g1,g2,g3,g4,sk,step,max:integer;
begin
step:= 6;
sk:= 0;

```

```

max:=100;
g1:=0;
while g1<max do      //пока g1<max, выполняем цикл
begin
  g2:=g1+step;      //задаем значение g2 как g1+step
  while g2<max do   //пока g2<max, выполняем цикл
  begin
    g3:=g2+step;    //задаем значение g3 как g2+step
    while g3<max do //пока g3<max, выполняем цикл
    begin
      g4:=g3+step;  //задаем значение g4 как g3+step
      while g4<max do //пока g4<max, выполняем цикл
      begin
        if (g2 - g1 > g3 - g2) and (g3 - g2 > g4 - g3) and (g1 + g2 > g3)
and (g1 + g2 < g4) then sk:=sk+1;
        //проверка условия; при выполнении - увеличение счетчика
        g4:=g4+step; //увеличение g4 на step
        end;
        g3:=g3+step; //увеличение g3 на step
        end;
        g2:=g2+step; //увеличение g2 на step
        end;
        g1:=g1+step; //увеличение g1 на step
        end;
        writeln(sk); //печатать результата
      end.

```

В последней PascalABC-реализации программы получился достаточно громоздкий код, причиной чему «ручное» регулирование начальных значений и шага изменения переменных $g1$, $g2$, $g3$ и $g4$. Но время выполнения данной программы соизмеримо со временем выполнения программы, написанной через цикл `for`.

В последней программной реализации (язык VBA) имя переменной `step` изменено на `step6`. Причиной тому – `step` в VBA является элементом оператора `for`, а потому не может быть использовано как имя переменной.

```

Sub goal()
Dim g1 As Integer    'описание переменных как целые числа
Dim g2 As Integer
Dim g3 As Integer
Dim g4 As Integer
Dim step6 As Integer
Dim sk As Integer
Dim max As Integer
step6 = 6            'присвоение значений
max = 100
sk = 0
'вложенные циклы для формирования наборов значений
For g1 = 0 To max Step step6

```

```

For g2 = g1 + step6 To max Step step6
  For g3 = g2 + step6 To max Step step6
    For g4 = g3 + step6 To max Step step6
      If (g2 - g1 > g3 - g2) And (g3 - g2 > g4 - g3) And (g1 +
g2 > g3) And (g1 + g2 < g4) Then sk = sk + 1
      Next g4
    Next g3
  Next g2
Next g1
MsgBox (sk)          'печатать результата
End Sub

```

Приведенные коды, или им подобные, являются ответом данной задачи.

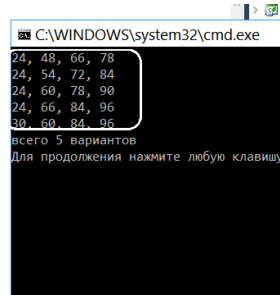
Интересно было бы также изучить оптимальность предложенных кодов и влияние исправления исходной блок схемы на скорость выполнения программы.

При выполнении любой из программ ответом будет 5, то есть существует пять вариантов забитых голов, при которых соблюдаются все условия задачи. Чтобы просмотреть возможные варианты следует изменить оператор if. После каждого найденного набора значений будем не только увеличивать значение sk, но и выведем данный набор на печать. Например, в C# это будет выглядеть так:

```

static void Main(string[] args)
{
    int g1, g2, g3, g4;
    int step = 6;
    int max = 100;
    int sk = 0;
    for (g1 = 0; g1 < max; g1 = g1 + step)
        for (g2 = g1 + step; g2 < max; g2 = g2 + step)
            for (g3 = g2 + step; g3 < max; g3 = g3 + step)
                for (g4 = g3 + step; g4 < max; g4 = g4 + step)
                    if ((g2 - g1 > g3 - g2) && (g3 - g2 > g4 - g3) &&
(g1 + g2) > g3 && (g1 + g2) < g4)
                        sk++;
    Console.WriteLine("{0}, {1}, {2}, {3}", g1, g2, g3, g4);
    Console.WriteLine("всего {0} вариантов", sk);
}

```



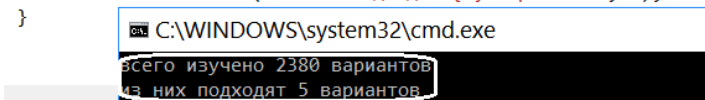
Итак, программа имеет 5 вариантов решений. При решении олимпиадной задачи в аудитории данный результат получить затруднительно. Этого и не требуется. Оценим, сколько всего было рассмотрено вариантов наборов значений. Для этого введем переменную-счетчик, которая будет увеличиваться на единицу при каждом выполнении оператора if; назовем ее count.

В приведенном скриншоте выделены внесенные изменения и результат. После всех возможных оптимизаций окончательная программа изучает 2380 вариантов, из них подходя только 5, что составляет 0,21 процента.

```

static void Main(string[] args)
{
    int g1, g2, g3, g4;
    int step = 6;
    int max = 100;
    int sk = 0;
    int count = 0;
    for (g1 = 0; g1 < max; g1 = g1 + step)
        for (g2 = g1 + step; g2 < max; g2 = g2 + step)
            for (g3 = g2 + step; g3 < max; g3 = g3 + step)
                for (g4 = g3 + step; g4 < max; g4 = g4 + step)
                {
                    count++;
                    if ((g2 - g1 > g3 - g2) && (g3 - g2 > g4 - g3) &&
                        (g1 + g2) > g3 && (g1 + g2) < g4)
                        sk++;
                }
    Console.WriteLine("всего изучено {0} вариантов", count);
    Console.WriteLine("из них подходят {0} вариантов", sk);
}

```

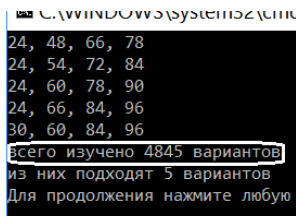


Если незначительно пренебречь аккуратностью и начальное проверяемое значение количества забитых голов следующим футболистом устанавливать равным количеству забитых голов текущего футболиста, то количество рассматриваемых вариантов будет 4845, то есть увеличится вдвое!

```

{
static void Main(string[] args)
{
    int g1, g2, g3, g4;
    int step = 6;
    int max = 100;
    int sk = 0;
    int count = 0;
    for (g1 = 0; g1 < max; g1 = g1 + step)
        for (g2 = g1; g2 < max; g2 = g2 + step)
            for (g3 = g2; g3 < max; g3 = g3 + step)
                for (g4 = g3; g4 < max; g4 = g4 + step)
                {
                    count++;
                    if ((g2 - g1 > g3 - g2) && (g3 - g2 > g4 - g3) &&
                        (g1 + g2) > g3 && (g1 + g2) < g4)
                    {
                        sk++;
                        Console.WriteLine("{0}, {1}, {2}, {3}", g1, g2, g3, g4);
                    }
                }
    Console.WriteLine("всего изучено {0} вариантов", count);
    Console.WriteLine("из них подходят {0} вариантов", sk);
}
}

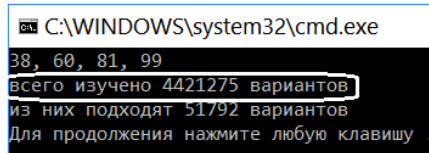
```



Здесь имеет место случай, когда ошибка в коде не влияет на результат: получившиеся варианты совпадают с правильными ответами, их по-прежнему всего пять. Но в подобном случае обязательно следовало бы вводить проверку на $g1 < g2 < g3 < g4$, что также замедлит общее время выполнения программы.

Напоследок рассмотрим, сколько вариантов пришлось бы рассмотреть, если бы шаг был бы равен единице:

```
{
    static void Main(string[] args)
    {
        int g1, g2, g3, g4;
        int step = 1;
        int max = 100;
        int sk = 0;
        int count = 0;
        for (g1 = 0; g1 < max; g1 = g1 + step)
            for (g2 = g1; g2 < max; g2 = g2 + step)
                for (g3 = g2; g3 < max; g3 = g3 + step)
                    for (g4 = (локальная переменная) int g3 + step)
                    {
                        count++;
                        if ((g2 - g1 > g3 - g2) && (g3 - g2 > g4 - g3) &&
                            (g1 + g2) > g3 && (g1 + g2) < g4)
                        {
                            sk++;
                            Console.WriteLine("{0}, {1}, {2}, {3}", g1, g2, g3, g4);
                        }
                    }
                Console.WriteLine("всего изучено {0} вариантов", count);
                Console.WriteLine("из них подходят {0} вариантов", sk);
    }
}
```



Как видно из скриншота, общее количество рассматриваемых вариантов составило 4421275. То есть почти в две тысячи раз больше, чем в оптимизированном варианте! При запуске программы время ее работы на компьютере средней мощности составило порядка 2 секунд. В скриншоте видно, что количество «подходящих» вариантов резко увеличилось: причиной тому – отсутствие проверки на кратность 2 и 3 и на неравенство переменных. Если добавить в программу эти проверки, время выполнения еще более увеличится; и код также усложнится.

Приведенные цифры наглядно показывают, что небольшие изменения исходной блок-схемы, полученные путем математических и логических умозаключений, реально позволяет ускорить время выполнения программ.

Типичные ошибки, допущенные при реализации задачи

1. Непонимание задачи, неумение ее решать.

2. Нахождение единственного (или нескольких) варианта количества забитых голов, при которых соблюдаются все условия задачи. Этого по сути задачи не требуется.

3. Составление неоптимального алгоритма, с пренебрежением условием «Все игроки забили разное количество голов, но количество забитых голов каждого делится без остатка и на 3 и на 2». Данная ошибка может позволить получить правильное число в результате работы программы, но данная задача проверяет не только навыки программирования и алгоритмизации, но и умение экономить машинные ресурсы.

Задача 12 (Олимпиада школьников МИСиС, 2010 год)

Написать оптимальную по времени выполнения программу или составить блок-схему алгоритма поиска наименьшего числа, которое, будучи разделено на 2, дает в остатке 1, при делении на 3, дает в остатке 2, при делении на 4, дает в остатке 3, при делении на 5, дает в остатке 4, при делении на 6, дает в остатке 5, но на 7 это число делится нацело.

Проверяется

Навыки алгоритмизации, математическая подготовка.

Обсуждается

На примере решения данной задачи проводится анализ времени выполнения программы, удачного и неудачного программного кода.

Помимо приведенной задачи даются общие рекомендации по решению подобных задач в общем виде.

Вводится понятие подпрограммы и дается пример ее использования.

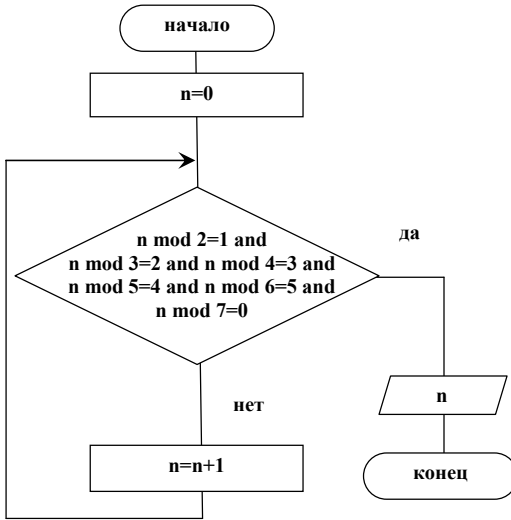
Вводится понятие рекуррентной функции и дается пример ее использования для решения данной задачи.

Теоретическое сопровождение темы. Решение

В первом приближении данная задача является весьма несложной и сводится к простому перебору всех целых чисел; для каждого числа проводится проверка; если проверка прошла успешно, число выводится на печать. Блок-схема описанного алгоритма и код на Pascal ABC приведены ниже.

При несомненной простоте данный алгоритм имеет весьма существенные недостатки:

1. Перебираются все значения чисел n . Простейших арифметических знаний достаточно, чтобы увидеть значения n , которые не подходят. Соответственно, нет смысла увеличивать количество проверяемых значений.



```

var n:integer;
begin
n:=0;
while not((n mod 2=1)
and (n mod 3=2 )and
(n mod 4=3 )and (n mod
5=4 )and (n mod 6=5
)and (n mod 7=0 )) do
n:=n+1;
writeln(n);
end.
  
```

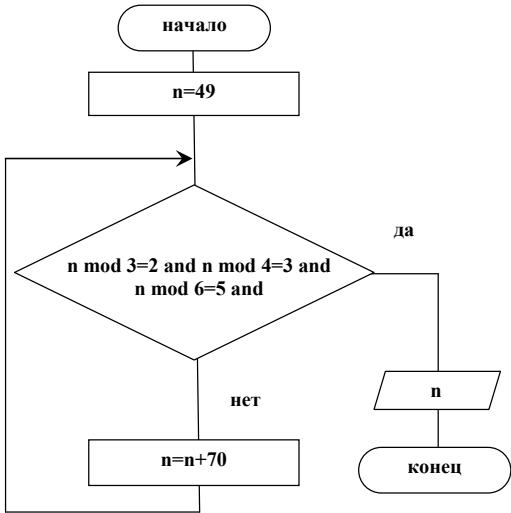
2. Для каждого числа n полностью производится проверка остатка: даже если первая часть составного условия $(n \bmod 2 = 1)$ не выполняется, все равно должны быть рассчитаны все прочие 5 условий. С позиции оптимальности времени выполнения такая программа очень неудачна.

Начнем с первого недостатка: если на 7 это число делится нацело, значит, данное число кратно 7. Таким образом, в алгоритме команду $n = n + 1$ можно изменить на $n = n + 7$. Если число, будучи разделено на 2, дает в остатке 1, следовательно, число нечетное; все четные множители семерки исключаются. В алгоритме команду $n = n + 7$ можно изменить на $n = n + 14$.

Обратите внимание, что даже при минимальных изменениях в алгоритме общая скорость выполнения программы будет увеличена на порядок (проверяется меньшее количество чисел).

Из дальнейшего наблюдения следует, что число, которое при делении на 5 дает в остатке 4, заканчивается либо на 4, либо на 9. Четным оно быть не может. Поэтому, число должно заканчиваться на 9. Из таблицы умножения следует, что наименьшее нечетное число, делящееся на 7 и заканчивающееся на 9 – это 49; следующее число – это $49 + 70 = 119$; следующее $119 + 70 = 189$, и т.д.

Измененный алгоритм представлен ниже. Обратите внимание, что в алгоритме за счет изменение начального значения n и шага изменения убраны проверки деления на 2, на 5 и на 7, что также увеличивает скорость выполнения программы.

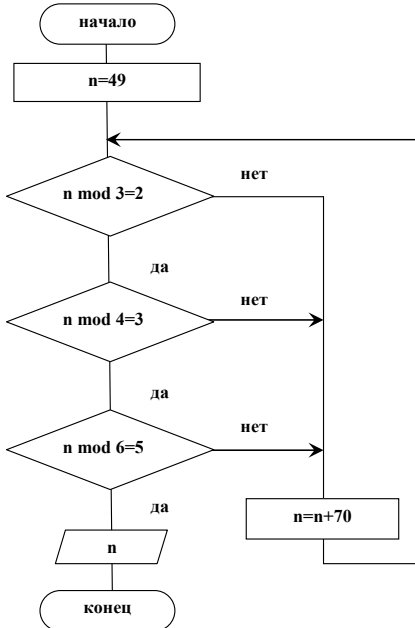


```

var n:integer;
begin
n:=49;
while not(
(n mod 3=2 )and
(n mod 4=3 )and
(n mod 6=5 )) do
n:=n+70;
writeln(n);
end.
  
```

Исправив, насколько это возможно, первый недостаток исходного кода, перейдём ко второму: потеря машинного времени на проверку последующих условий, если не выполняется предыдущее.

Проверять условия будем последовательно. Алгоритм приведен ниже.



```

var n:integer;
b:boolean;
begin
n:=-21;
b:=true;
while b do
begin
n:=n+70;
if n mod 3=2 then
if n mod 4 =3 then
if n mod 6=5 then
b:=false;
end;
writeln(n);
end.
  
```


Стоит обратить внимание на реализацию алгоритма в программном виде: программный код отличается от изображенной блок-схемы. Если реализовывать программу в полном соответствии с блок-схемой, то потребуется многократное введение рассмотренных в предыдущей задаче, операторов goto. В данном коде для обеспечения повторения нужных действий применен оператор цикла по условию while.

В программный код введена дополнительная булевская переменная, принимающая значение true/false (Истина/Ложь), которая выполняет роль переключателя: при обнаружении первого требуемого числа значение переменной меняется на противоположенное, что блокирует дальнейшее вхождение в цикл по условию.

Также в программе для обеспечения верности первого проверяемого значения исходное значение переменной n, заданное до цикла, равно $49 - 70 = -21$.

Ответ в данной задаче – число 119.

Можно увидеть, приведенный алгоритм исправил все указанные недостатки. Но, в результате сокращения времени выполнения программы, был несколько усложнен программный код:

1. Введена дополнительная переменная, что является нормальной ситуацией в программировании.

2. Введено несколько проверок (if), что для данной задачи оправдано и не столь критично, но не всегда подобный код с множественными одностипными проверками является удачным решением.

Рассмотрим подробнее второй пункт: если какое-то одностипное действие повторяется некоторое количество раз, в программировании рекомендуется использовать циклы. При использовании цикла алгоритм будет модифицирован следующим образом:

Без цикла с множественным if

```
var n:integer;
b:boolean;
begin
n:=-21;
b:=true;
while b do
begin
n:=n+70;
if n mod 3=2 then
if n mod 4 =3 then
if n mod 6=5 then
b:=false;
end;
writeln(n);
end.
```

С использованием цикла

```
var n,i,sk:integer;
b:boolean;
begin
n:=-21;
b:=true;
while b do
begin
n:=n+70;
sk:=0;
for i:=3 to 6 do
if n mod i<>i-1 then break
else sk:=sk+1;
if sk=4 then b:=false;
end;
writeln(n);
end.
```

Использование цикла при решении данной задачи является несколько спорным преимуществом. С одной стороны, код становится более универсальным, с другой стороны, при использовании цикла код усложняется:

- вводится дополнительная переменная, которая подсчитывает количество успешно пройденных проверок; если все проверки пройдены успешно – значит, число найдено.
- проводится лишний шаг проверки деления на 5 с остатком 4, который в данной задаче учтен при формировании исходных данных и определении шага изменения переменной *n*.

Первый недостаток не всегда является критичным: в VBA и C# переменную *sk* вводить не требуется. В этих языках при выполнении цикла *for* переменная цикла сначала увеличивается на заданный шаг, а далее анализируется ограничивающее условие, определяющее возможно ли входить в цикл повторно. Таким образом, при успешном завершении цикла значение переменной цикла будет равно 7. Ниже приводится вариант кода задачи для различных языков программирования.

Pascal ABC	C#	VBA
<code>var n,i,sk:integer;</code>	<code>static void Main(string[] args)</code>	<code>Sub task()</code>
<code>b:boolean;</code>	<code>{</code>	<code>n = -21</code>
<code>begin</code>	<code> int i;</code>	<code>b = True</code>
<code>n:=-21;</code>	<code> int n = -21;</code>	<code>While b</code>
<code>b:=true;</code>	<code> bool b=true;</code>	<code> n = n + 70</code>
<code>while b do</code>	<code> while (b)</code>	<code> For i = 3 To 6</code>
<code>begin</code>	<code> {</code>	<code> If n Mod i <> i - 1</code>
<code>n:=n+70;</code>	<code> n=n+70;</code>	<code> Then Exit For</code>
<code>sk:=0;</code>	<code> for (i = 3; i < 7; i++)</code>	<code> Next i</code>
<code>for i:=3 to 6 do</code>	<code> if (n % i != i - 1) break;</code>	<code> If i = 7 Then b =</code>
<code>if n mod i<>i-1</code>	<code> if (i==7) b=false;</code>	<code> False</code>
<code>then break</code>	<code> }</code>	<code> Wend</code>
<code>else sk:=sk+1;</code>	<code> Console.WriteLine(n);</code>	<code> MsgBox (n)</code>
<code>if sk=4 then</code>	<code> }</code>	<code>End Sub</code>
<code>b:=false;</code>		
<code>end;</code>		
<code>writeln(n);</code>		
<code>end.</code>		

Второй недостаток не зависит от языка программирования. Однако при решении задач, в которых производятся однотипные действия некоторое количество раз, стоит иметь в виду, что алгоритм с циклом более предпочтительный.

Усложним текущую задачу:

«Найти число, которое, будучи разделено на 2, дает в остатке 1, при делении на 3, дает в остатке 2, при делении на 4, дает в остатке 3, при делении на 5, дает в остатке 4, при делении на 6, дает в остатке 5,

при делении на 7, дает в остатке 6, при делении на 8, дает в остатке 7, при делении на 9, дает в остатке 8, при делении на 10, дает в остатке 9, но на 11 это число делится нацело».

Для решения усложненной задачи в алгоритм, не использующий цикл, придется вводить значительные изменения, увеличивая количество проверок. Представьте, насколько возрастет сложность алгоритма, если последним числом будет не 11, а, например, 29!

Алгоритм с циклом модифицируется проще:

```
var n,i,sk:integer;
b:boolean;
begin
n:=0;
b:=true;
while b do
begin
n:=n+11;
sk:=0;
for i:=3 to 10 do
if n mod i<>i-1 then break
else sk:=sk+1;
if sk=8 then b:=false;
end;
writeln(n);
end.
```

Для создания как можно более универсального кода, изменено начальное значение переменной n (n=0), шаг параметра и количество проверяемых делителей.

В глобальном плане предпочтительнее писать как можно более универсальные коды программ, которые при изменении входных данных могут быть легко трансформированы под новые условия. Применительно к последней версии алгоритма, его можно переписать так, чтобы можно было использовать один и тот же алгоритм для решения задачи общего вида:

«Найти число, которое,

- будучи разделено на 2, дает в остатке 1,
- при делении на 3, дает в остатке 2,
- при делении на 4, дает в остатке 3,
- ...
- при делении на k-1, дает в остатке k-2,
- но на k это число делится нацело»

Как видно из предыдущего алгоритма, значение k приводит к изменениям в значении шага и в количестве проверок. Для того, чтобы

программа работала при любом k, вводим переменную k, которая используется вместо числовых констант.

Pascal	<pre> var n, i, sk, k: integer; ABC b: boolean; begin readln(k); n:=0; b:=true; while b do begin n:=n+k; sk:=0; for i:=2 to k-1 do if n mod i <> i-1 then break else sk:=sk+1; if sk=k-2 then b:=false; end; writeln(n); end.</pre>	<pre> ‘описание переменных ‘ ‘запрос k ‘ ‘ ‘ ‘ ‘ ‘печать результата</pre>
C#	<pre> static void Main(string[] args) { int i; int n = 0; bool b=true; int k = int.Parse(Console.ReadLine()); while (b) { n=n+k; for (i = 3; i < k; i++) if (n % i != i - 1) break; if (i==k) b=false; } Console.WriteLine(n); }</pre>	<pre> // //описание и инициализация //переменных // //запрос k и преобразование в // число // // //печать результата</pre>
VBA	<pre> Sub task() k = CInt(InputBox("Введите число")) n = 0 b = True While b n = n + k For i = 2 To k - 1 If n Mod i <> i - 1 Then Exit For Next i If i = k Then b = False Wend MsgBox (n) End Sub</pre>	<pre> ‘описание ‘переменных ‘запрос k ‘ ‘ ‘ ‘ ‘печать результата</pre>

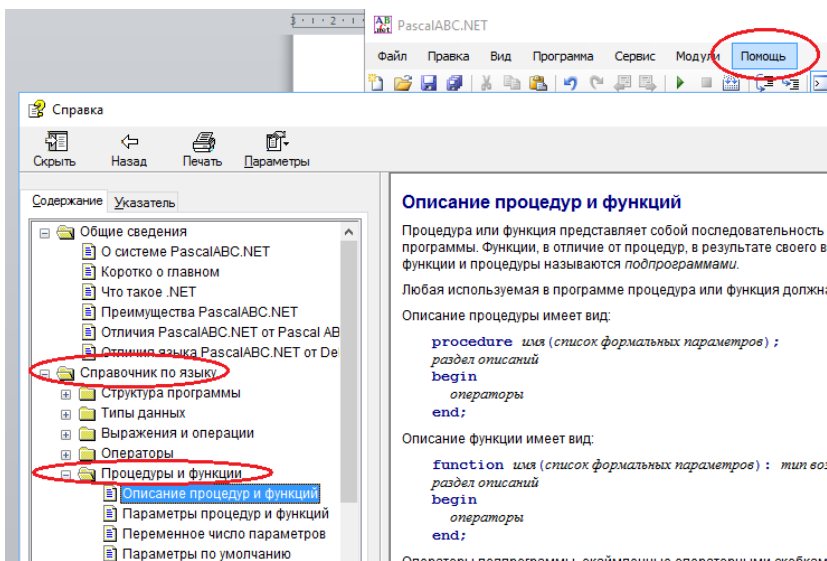
Примечание: детальный анализ приведенных кодов показывает, что не при любых значениях переменной k программа будет благополучно завершена. Вопрос допустимых значений переменной k предлагаем, как дополнительное задание.

В рамках обсуждения данной задачи представляется удобным ввести понятие подпрограммы и целесообразность ее применения.

В рамках школьного курса подпрограммам не всегда уделяется достаточно внимания, в то время, как большинство реальных программ строится именно из законченных программных модулей, выполняющих некоторую подзадачу.

В различных языках программирования вводятся различные термины для обозначения этих модулей – подпрограмма, макрос, функция, процедура, метод – все это слова синонимы, хотя надо знать, что между ними существуют некоторые принципиальные различия.

В данном пособии не ставится целью дать детальные инструкции применения подпрограмм. Эту информацию можно почерпнуть из справочников или из справки по языку. Ниже показан для Pascal ABC.



Целью ставится продемонстрировать применение и обосновать необходимость использования процедур и функций в Pascal ABC, макросов с параметром и функций в VBA, методов в C#.

Для наглядности необходимости и удобства использования подпрограмм решим задачу общего вида для $k = 7$, $k = 11$, $k = 13$.

Очевидно, что, задание может быть выполнено трехкратным выполнением существующего кода, при вводе требуемых значений k . Но, данный способ решения задачи не всегда удобен, так как требует участия оператора, вводящего числовые значения. В некоторых слу-

чаях требуется, что программный код работал автоматически. И, что более важно, данный способ не сохраняет полученные результаты для дальнейшей работы (например, для сравнения).

Конечно, можно, изменив ввод переменной *k* с клавиатуры на задание конкретного значения, повторить в рамках одной программы трижды для разных значений *k*. Данный вариант в программировании считается неудачным:

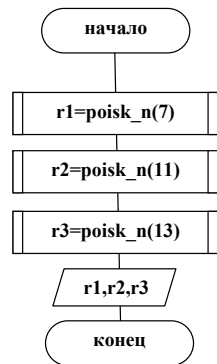
- необоснованно длинный код,
- в случае редактирования, изменения надо будет повторять несколько раз, что повышает вероятность ошибки.

Альтернативным вариантом будет преобразование уже существующей программы в подпрограмму, в частности – в функцию `poisk_n` (для C# – это метод). Функция в программировании, как и функция в математике, имеет некоторый набор исходных параметров, указываемых в скобках после имени функции, и возвращает в результате своего приенения единственное значение.

Итак, ниже приведен код функции `poisk_n`, которая получает на вход значение *k*, а в качестве результата возвращает найденное значение *n*. Эта функция будет вызываться из основной программы несколько раз с разными значениями *k*. Код основной программы сводится к вызову функции с разными параметрами.

```
Pascal function poisk_n(k:integer):integer;  
ABC var n,i,sk:integer;  
      b:boolean;  
      begin  
        n:=0;  
        b:=true;  
        while b do  
          begin  
            n:=n+k;  
            sk:=0;  
            for i:=2 to k-1 do  
              if n mod i<>i-1 then break  
              else sk:=sk+1;  
              if sk=k-2 then b:=false;  
            end;  
          poisk_n:=n;  
        end;  
  
      var r1,r2,r3:integer;  
      begin  
        r1:=poisk_n(7);  
        writeln(r1);  
        r2:=poisk_n(11);  
        writeln(r2);  
        r3:=poisk_n(13);  
        writeln(r3);  
      end.
```

Блок-схема алгоритма



```

C#      class Program
        {
            static void Main(string[] args)
            {
                int r1 = poisk_n(7);
                Console.WriteLine(r1);
                int r2 = poisk_n(11);
                Console.WriteLine(r2);
                int r3 = poisk_n(13);
                Console.WriteLine(r3);
            }
            static int poisk_n(int k)
            {
                int i;
                int n = 0;
                bool b = true;
                while (b)
                {
                    n = n + k;
                    for (i = 3; i < k; i++)
                        if (n % i != i - 1) break;
                    if (i == k) b = false;
                }
                return n;
            }
        }
}

```

```

VBA      Function poisk_n(k)
n = 0
b = True
While b
    n = n + k
    For i = 2 To k - 1
        If n Mod i <> i - 1 Then Exit For
    Next i
    If i = k Then b = False
Wend
poisk_n = n
End Function

Sub task_new()
r1 = poisk_n(7)
MsgBox (r1)
r2 = poisk_n(11)
MsgBox (r2)
r3 = poisk_n(13)
MsgBox (r3)
End Sub

```

Примечание: основным отличием функций от процедур является возврат единственного значения для заданного набора входных параметров (аналогично математическим функциям). Процедура, в отличие от функции, может совсем не возвращать значения (или возвращать несколько результатов).

Естественно, при решении задач необязательно применять вызов подпрограмм, но в целом модульный принцип создания программы

- структурирует код,
- улучшает восприятие,
- сокращает текст программы.

Получив представление о применении функций, перейдем к понятию рекуррентной функции. Рекуррентной является функция, которая в процессе выполнения вызывает сама себя. В общем виде, рекуррентная функция будет выглядеть следующим образом:

```
function recurrent (...):...;
begin
    recurrent ();
end;
```

Естественно, вызов функции из функции без ограничений, приводит к закликиванию программы. Поэтому, как правило, в теле функции имеется условие, по выполнению которого производится либо повторный вызов данной функции, либо завершение работы функции:

```
function recurrent (...):...;
begin
    if (<условие>) then recurrent() else recurrent=...;
end;
```

Классическим примером применения рекуррентной функции является задача расчета факториала $n! = 1 * 1 * 2 * 3 * \dots * n$.

Pascal ABC	C#	VBA
<pre>function f(k:integer):integer; begin if k<>1 then f:=k*f(k-1) else f:=1; end; var r1:integer; begin r1:=f(6); writeln(r1); end.</pre>	<pre>class Program { static void Main(string[] args) { int r1 = f(6); Console.WriteLine(r1); } static int f(int k) { if (k != 1) return k*f(k - 1); else return 1; } }</pre>	<pre>Function f(k As Integer) If k <> 1 Then f = k * f(k - 1) Else f = 1 End If End Function Sub task() res = f(6) MsgBox (res) End Sub</pre>

Рассмотрим приведённый пример подробнее. Основная программа для вычисления, например 6!, вызывает функцию f с параметром 6: f(6).

При реализации функции f использовано, что $n! = 1 * 1 * 2 * 3 * \dots * n$ можно вычислить как $n! = n * (n-1) * (n-2) * \dots * 1$: от перестановки мест множителей произведение не меняется. То есть получается, что

$$n! = n * (n-1) * (n-2) * \dots * 1 = n * (n-1)!$$

Этот математический вывод использован в реализации логики работы функции f : $f(n) = n * f(n-1)$. То есть функция вызывает сама себя, при каждом следующем вызове передавая меньший параметр до тех пор, пока передаваемый параметр не станет равен единице.

В частности, при вычислении $f(6)$ производится следующая цепочка действий:

- вызывается функция f с параметром 6: параметр отличен от единицы; следовательно, результат работы функции должен быть рассчитан как $6 * f(6-1)$ (**if** $k < 1$ **then** $f := k * f(k-1)$). Работа функции $f(6)$ не завершена: для дальнейших расчетов требуется вызвать функцию f с параметром 5;

- вызывается функция f с параметром 5: параметр отличен от единицы; следовательно, результат работы функции должен быть рассчитан как $5 * f(5-1)$. Работа функции $f(5)$ не завершена: для дальнейших расчетов требуется вызвать функцию f с параметром 4;

- ...

- вызывается функция f с параметром 2; параметр отличен от единицы; следовательно, результат работы функции должен быть рассчитан как $2 * f(2-1)$. Работа функции $f(2)$ не завершена: для дальнейших расчетов требуется вызвать функцию f с параметром 1;

- вызывается функция f с параметром 1: параметр равен единице; следовательно, результат работы функции должен быть равен 1 (**else** $f := 1$);. Работа функции $f(1)$ завершена: результат ее работы - 1 – передается в функцию $f(2)$;

- продолжается работа функции $f(2)$: $2 * f(2-1) = 2 * 1 = 2$; результат работы $f(2)$ – 2 – передается в функцию $f(3)$;

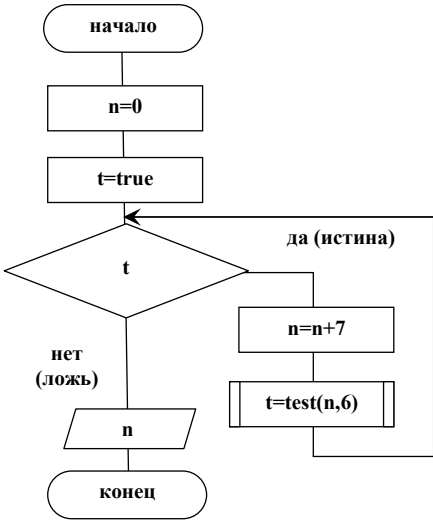
- продолжается работа функции $f(3)$: $3 * f(3-1) = 3 * 2 = 6$; результат работы $f(3)$ – 6 – передается в функцию $f(4)$;

- ...

- продолжается работа функции $f(6)$: $6 * f(6-1) = 6 * 20 = 120$; результат работы $f(6)$ передается в основную программу.

В завершение описания вариантов решения поставленной задачи представляется интересным предложить вариант решения исходной задачи с использованием рекуррентной функции. Введем рекуррентную функцию, которая будет проверять остаток от деления проверяемого числа на делитель; при успешной проверке данная функция будет вызывать сама себя повторно, передавая в качестве параметра следующий делитель. При первой неуспешной проверке рабо-

та функции завершается. Тип возвращаемого значения – булевский. Основная программа увеличивает значение n до тех пор, пока функция не возвратит значение False.

<pre> Pascal ABC function test(n,k:integer) : boolean; begin if (k=1) then test:=false else if (n mod k=k-1) then test:=test(n,k-1) else test:=true; end; var b:boolean; n:integer; begin n:=0; b:=true; while b do begin n:=n+7; b:=test(n,6); end; writeln(n); end. </pre>	<pre> C# class Program { static void Main(string[] args) { int n = 0; bool b=true; while (b) { n=n+7; b=test(n, 6); } Console.WriteLine(n); } static bool test(int n, int k) { if (k == 1) return false; else { if (n % k == k - 1) return test(n, k - 1); else return true; } } } </pre>
<pre> VBA Sub task () n = 0 b = True While b n = n + 7 b = test(n, 6) Wend MsgBox (n) End Sub Function test(n, k) If k = 1 Then test = False Else If n Mod k = k - 1 Then test = test(n, k - 1) Else test = True End If End If End Function </pre>	 <pre> graph TD Start([начало]) --> N0[n=0] N0 --> Ttrue[t=true] Ttrue --> T{t} T -- да (истина) --> Nplus7[n=n+7] Nplus7 --> Ttest[t=test(n,6)] Ttest --> T T -- нет (ложь) --> N[/n/] N --> End([конец]) </pre>

Типичные ошибки, допущенные при реализации задачи

1. Математические просчеты.
2. Неверное понимание оптимального алгоритма.

Заключение

В данном пособии ко всем заданиям приведены решения и детально разобраны программные коды. Читателю рекомендуют не только теоретически изучить приведенные задачи, но и самостоятельно повторить программную реализацию задач.

Для практического закрепления навыков авторы рекомендуют самостоятельно реализовать программные коды задач 2,3 и 5. Несмотря на то, что в приведенных заданиях программная реализация не предусмотрена, эти задачи дают возможность попрактиковаться в реализации логики алгоритма, в работе с массивами и с тестовыми данными.

И, наконец, последний совет участникам олимпиад. «МИСиС зажигает звезды» предполагает решение заданий вне дисплейного класса. При этом разрешается приводить в качестве решения программы, написанные на любом языке программирования. Во избежание спорных ситуаций советуем приводить комментарии к программе. Особо актуальна данная рекомендация при использовании мало распространенных языков программирования (к широко распространенным языкам авторы относят любой Basic, любой Pascal, любой C).

Учебное издание

КРЫНЕЦКАЯ Галина Сергеевна
СИДОРОВ Сергей Васильевич

**Методическое пособие по подготовке к олимпиадам
школьников инженерной направленности**

**Информационно-технологическое направление
«Математика в информатике.
Решение олимпиадных задач»**

7–8-й классы

В авторской редакции

Подписано в печать 06.12.17	Бумага офсетная	Уч.-изд. л. 4,25
Формат 60 × 90 ¹ / ₁₆	Печать офсетная	Заказ

Национальный исследовательский
технологический университет «МИСиС»,
119049, Москва, Ленинский пр-т, 4

Издательский Дом МИСиС,
119049, Москва, Ленинский пр-т, 4
Тел. (495) 638-45-22

Отпечатано в типографии Издательского Дома МИСиС
119049, Москва, Ленинский пр-т, 4
Тел. (499) 236-76-17, тел./факс (499) 236-76-35