МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

С.В. Сидоров, Г.С. Крынецкая

ИНФОРМАТИКА

МЕТОДИЧЕСКОЕ ПОСОБИЕ ПО ПОДГОТОВКЕ К ОЛИМПИАДАМ ШКОЛЬНИКОВ

9–11-й классы

Рекомендовано редакционно-издательским советом

МИСиС

Москва 2016

УДК 681.3
С34

Сидоров С.В.

С34 Информатика : метод. пособие по подготовке к олимпиадам школьников : 9–11-й классы / С.В. Сидоров, Г.С. Крынецкая. – М. : Изд. Дом МИСиС, 2016. – 74 с.

Цель данного пособия – помочь школьникам эффективно подготовиться к олимпиадам по информатике.

Пособие содержит примеры олимпиадных заданий с разбором, анализ их выполнения учащимися, а также справочные материалы, охватывающие все представленные на олимпиаде разделы информатики.

Пособие предназначено для школьников 6–11 классов и для учителей информатики. Материалы пособия могут быть использованы для подготовки к различным олимпиадам по информатике, а также на уроках информатики.

УДК 681.3

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время мы наблюдаем стремительное развитие информационных технологий и их применение в самых различных отраслях человеческой деятельности. 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 создания и отладки коды программ, более глубокому изучению языков программирования, умению работать с символьными и с текстовыми типами данных.

Олимпиада по информатике «МИСиС зажигает звезды» проводится для 9, 10 и 11 классов в два этапа: отборочный и заключительный. Варианты для 9,10 и 11 класса 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:

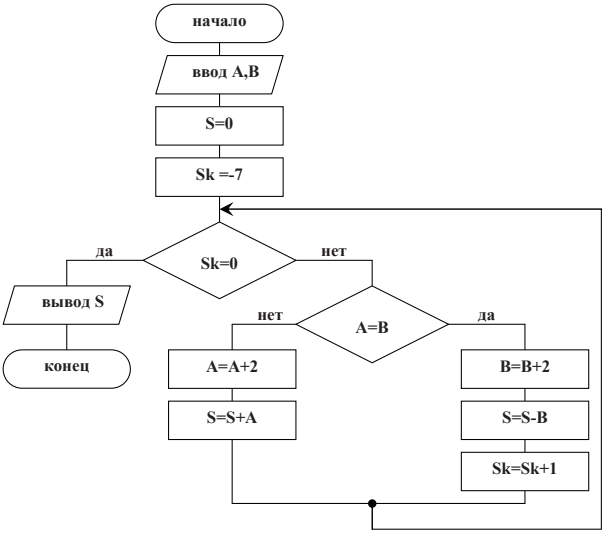
- Pascal ABC, как наиболее популярный для изучения в школах;
- C# по причине углубленного изучения в МИСиС;
- VBA (Visual Basic for Application) как наиболее распространенная версия Бейсика, входящая в состав пакета Microsoft Office.

Это позволяет наиболее 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 9, 10 и 11 классов. Также оно может быть полезно студентам первых курсов и учителям информатики.

Задача 1 (Олимпиада школьников МИСиС, 2010 год)

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



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 A и B, устанавливается значение счетчика Sk=-7. Значения переменных A и B 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 A и B. Следовательно, для успешного завершения программы необходимо, чтобы значения переменных регулярно совпадали.

Если A исходно будет больше B, то выполнение программы приведет к дальнейшему увеличению A, следовательно, значения переменных уже не совпадут, счетчик не увеличится, произойдёт зацикливание алгоритма. Следовательно, исходное значение A не должно быть больше B.

При A равном B, алгоритм завершится благополучно. Следовательно, A может быть равно B.

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

Получается, что A=B, либо A<B: A+k=B, k – четное целое больше нуля. Эти два условия можно объединить в одно. Окончательный ответ: Алгоритм успешно закончит работу при A+k=B, k – четное целое >=0

Типичные ошибки, допущенные при реализации задачи

1. Непонимание алгоритма
2. Нахождение единственного варианта ответа A=B

Задача 2 (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 a) и d) получаются больше 100_{10}. В пункте e) задано число на единицу меньшее, чем $1000\ 0000_2$. Следовательно, e) – также больше 100_{10}.

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

Следовательно, ответ

Больше100_{10}.: $1000\ 0001_2$, , $1110\ 1001_2$, $0111\ 1111_2$,

Меньше 100_{10}.: $0110\ 000_2$, $0010\ 1001_2$, $0100\ 0011_2$,

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 3 (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; исходно задано 2 угла; надо вычислить третий;

• должны отсутствовать математические ошибки;

• рекомендуется демонстрация четкого алгоритмического мышления;

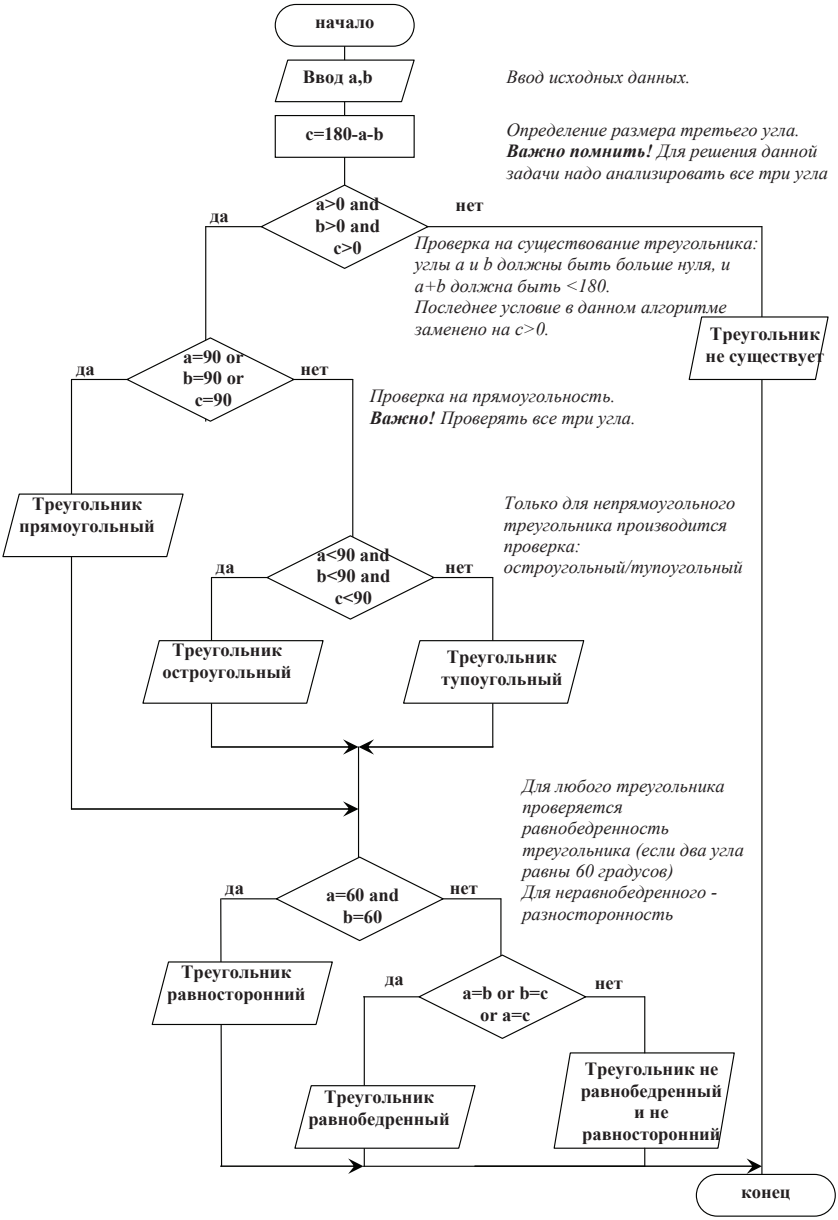
• учитывается лаконичность алгоритма;

• учитывается грамотное построение блок-схемы. Естественно, построение алгоритма – творческий процесс, но для обеспечения понятности составленной блок-схемы важно придерживаться существующих стандартов:

– точка начала и окончания алгоритма отображаются овалом

– ввод/вывод данных отображается параллелограммом

– выполнение операций отображается прямоугольником; причем в одном прямоугольнике можно перечислить несколько действий


начало
Ввод a,b
Ввод исходных данных.
c=180-a-b
Определение размера третьего угла.
Важно помнить! Для решения данной задачи надо анализировать все три угла
a>0 and b>0 and c>0
да
нет
Проверка на существование треугольника: углы a и b должны быть больше нуля, и a+b должна быть <180.
Последнее условие в данном алгоритме заменено на c>0.
Треугольник не существует
a=90 or b=90 or c=90
да
нет
Проверка на прямоугольность.
Важно! Проверять все три угла.
Треугольник прямоугольный
Только для непрямоугольного треугольника производится проверка:
остроугольный/тупоугольный
a<90 and b<90 and c<90
да
нет
Треугольник остроугольный
Треугольник тупоугольный
Для любого треугольника проверяется равнобедренность треугольника (если два угла равны 60 градусов)
Для неравнобедренного - разносторонность
a=60 and b=60
да
нет
Треугольник равносторонний
a=b or b=c or a=c
да
нет
Треугольник равнобедренный
Треугольник не равнобедренный и не равносторонний
конец

– 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 выше 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 равносторонность. 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 тремя 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 4 (10 класс, Роботрон, отборочный тур 2016 год)

Сколько значащих нулей в двоичной записи шестнадцатеричного числа 12AC0? (Ноль называется значащим, если удаление его из записи числа ведёт к изменению значения числа). Приведите решение задачи.

Напишите алгоритм (возможно в виде блок-схемы) определения количества значащих нулей в двоичной записи числа.

Проверяется

При решении данной задачи требуется продемонстрировать

• знание систем счисления, в частности, сопоставления двоичной системы с шестнадцатеричной;

• математические навыки,

• знание типов данных,

• способность определения типа обрабатываемых данных,

• умение работать со строковыми переменными

• навыки грамотного составления алгоритма для решения задачи.

Обсуждается

Шестнадцатеричная система счисления, соответствие между системами счисления в случае, если основание одной системы счисления является степенью другой.

Кодирование текстовой информации. Простейшие операции с текстовыми переменными.

Приемы программирования; целесообразность использования оператора goto и способы написания альтернативного кода без использования goto.

Теоретическое сопровождение темы. Решение

Первая часть задачи проверяет знание соответствия шестнадцатеричной и двоичной систем счисления. Естественно, можно перевести заданное шестнадцатеричное число в десятичное, а далее – в двоичное, но этот способ решения потребует слишком много времени и сопряжен с большим количеством ошибок расчета.

Для быстрого и легкого решения первой части задачи надо знать, что двоичная система счисления в настоящее время представляет особую важность по причине того, что все данные в современных вычислительных системах представляются в двоичном коде. Двоичный код, в свою очередь, обусловлен физической реализацией вычислительных систем (заряд есть/ заряда нет; транзистор открыт/ транзистор закрыт). Но двоичное представление информации неудобно для человеческого восприятия. Для удобства двоичные данные преобразуются в десятичные.

Классический пример: IP-адрес 4-ой версии протокола TCP/IP. IP-адрес представляет собой 32-х битное число, которое для представления пользователю разбивается на 4 октета, и каждый октет записы-

вается в десятичном виде. Например: IP-адрес 11000000 10101000 00001100 00010011 пользователю представляется как 192.168.12.19.

Но преобразование двоичного кода в десятичный и обратное ему не всегда очевидно. В приведенном примере октет 10101000 преобразуется в 168, но моментально это не вычисляется. Для чисел большей разрядности это утверждение еще более справедливо.
Таким образом, отображение двоичной системы удачнее производить в шестнадцатеричную систему, используя тот факт, что $16=2^4$.

шестнадцатеричный символ	соответствующее двоичное число	соответствующее десятичное число
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Продолжая пример IP-адресации вспомним, что IP-адрес 6-ой версии протокола TCP/IP представляет 128-ми битное число, которое отображается пользователю уже в шестнадцатеричной системе счисления. Например, 2A00:1370:811B:19D2:D213:8EC3:A456:8648. То есть 32 шестнадцатеричных символа, каждый из которых кодирует 4 символа двоичной системы. 32*4=128.

Очевидно, что для 6-ой версии TCP/IP представлять IP/адрес в десятичной системе было бы крайне неудобно.

Таким образом, возвращаясь к решению данной задачи, самым простым способом является прямой перевод заданного числа согласно указанной выше таблице из шестнадцатеричной системы в двоичную.

$12AC0_{16} = 0001\ 0010\ 1010\ 1100\ 0000_2$

Далее требуется продемонстрировать понятие значащего нуля: все нули, расположенные левее первого ненулевого символа, не меняют значения числа и не являются значащими.

$0001\ 0010\ 1010\ 1100\ 0000_2 = 1\ 0010\ 1010\ 1100\ 0000_2$.

Количество нулей справа подсчитывается вручную: данное число содержит 11 значащих нулей.

Вторая часть задания предполагает алгоритмизацию подсчета значащих нулей введенного в программу двоичного кода.

При реализации алгоритма важно понимать, что вводимая информация (то есть двоичный код числа) вводиться как строка. То есть переменная, содержащая исходные данные будет иметь тип string. Причин тому несколько:

• В противном случае (при вводе исходных данных как числа) задача частично теряет свой смысл: все незначащие нули отбрасываются автоматически.

• Часть языков программирования высокого уровня (например, C#) воспринимает вводимый с клавиатуры набор символов именно как строку.

• Вводимое число автоматически воспринимается как число десятичной системы счисления, и дальнейшие математические действия будут алгоритмически не вполне корректными по отношению к введенной информации.

Таким образом, в данной задаче требуется демонстрация понимания типов данных и работы со строковыми данными.

Теоретические аспекты работы с тестом следующие.

Для ввода, хранения и обработки текстовой информации предусмотрен свой способ представления информации в двоичном коде: любой символ, вводимый с клавиатуры, представляется соответствующим этому символу двоичным кодом в соответствии с таблицей кодирования.

Первой и наиболее известной таблицей кодирования была ASCII. Исходная таблица ASCII позволяла кодировать всего 128 символов, среди которых были буквы английского алфавита, цифры и все прочие клавиатурные символы. Преимуществом ASCII является малый размер кода для хранения символа. Недостатком – малое количество

представляемых символов, не позволяющее охватывать все алфавиты всех стран.

Наиболее распространенная в настоящее время таблица кодировок UNICODE. UNICODE имеет несколько стандартов, согласно которым для хранения одного символа отводится 2 или 4 байта.

Приведем примеры кодирования некоторых символов в ASCII и UNICODE.

символ	в кодировке ASCII	в кодировке UNICODE двоичный код	в кодировке UNICODE шестнадцатеричный код
A	0100 0001	0000 0000 0100 0001	0041
B	0100 0010	0000 0000 0100 0010	0042
C	0100 0011	0000 0000 0100 0011	0043
a	0110 0001	0000 0000 0110 0001	0061
b	0110 0010	0000 0000 0110 0010	0062
c	0110 0011	0000 0000 0110 0011	0063
А	1100 0000	0000 0100 0001 0000	0410
Б	1100 0001	0000 0100 0001 0001	0411
В	1100 0010	0000 0100 0001 0010	0412
а	1110 0000	0000 0100 0011 0000	0430
б	1110 0001	0000 0100 0011 0001	0431
в	1110 0010	0000 0100 0011 0011	0432
4	0011 0100	0000 0000 0011 0100	0034
5	0011 0101	0000 0000 0011 0101	0035
6	0011 0110	0000 0000 0011 0110	0036

Из детального рассмотрения фрагментов таблиц можно сделать следующие выводы:

• Имеет место частичное совпадение в кодах ASCII и UNICODE (в младших разрядах). В приведенной таблице это хорошо видно для символов латинского алфавита и для чисел. Но это не обеспечивает совместимости кодировок: для русского алфавита кодировки имеют различия.

• Символы представления заглавных и прописных букв различаются.

• В обеих кодировках двоичный код соответствует алфавитному порядку буквы.

• Символы, отображающие цифры, также представлены в таблицах кодировки. И хотя имеется некоторое соответствие между двоичной записью числа и его кодом в таблицах ASCII и UNICODE, важно понимать, что представление числа как числа и числа как символа различаются. Например, число «6», представленное в памяти вычислительной системы как число, будет иметь вид «0000 0110» (если предположить, что на число отводится 1 байт). Символ «6» будет записан в кодировке ASCII как «0011 0110» и в кодировке UNICODE «0000 0000 0011 0110».

Продолжая тему представления чисел как строкой информации, надо понимать, что число «66», записанное как число, будет иметь вид «0100 0010» (если предположить, что на число отводится 1 байт). Записанное как текст, число «66» будет представлено как набор (массив) символов «6» и «6». В кодировке ASCII будет записано как «0011 0110 0011 0110»; в кодировке UNICODE – «0000 0000 0011 0110 0000 0000 0011 0110».

В языках программирования для работы с текстом вводятся специальные переменные типов char и string. Переменная типа char содержит всегда только один символ. Переменная типа string содержит последовательность символов.

С каждым типом переменных предусмотрены свои допустимые операции.

Если для переменных числового типа допустимы операции сложения, вычитания, умножения, деления и проч. (набор математических операций различается в различных языках программирования), то для строковых переменных возможна операция сложения, определения длины строки, поиска подстроки в строке.

При этом операция сложения будет выполняться иначе, нежели для числовых данных. Например, для числовых переменных «123» и «654» операция сложения даст «777». Для строковых переменных «123» и «654» операция сложения даст «123654».

Операция определения длины строки для «123654» даст «6», операция поиска подстроки «6» в строке «123654» вернет позицию «4» (если нумерация символов в строке начинается с 1).

Возвращаясь непосредственно к решению задачи, исходим из того, что на вход программы поступает строка 0001 0010 1010 1100 0000, представляющая двоичный код шестнадцатеричного числа. (Естественно, программу можно алгоритмизировать полностью, вве-

дя исходно шестнадцатеричное число, и, преобразовав его в дальнейшем в двоичное представление. Но этот вариант значительно усложнит задачу.)

Основные шаги по решению задачи следующие:

1. Ввод данных

2. Анализ длины строки: если строка нулевая, то дальнейшее выполнение программы не требуется, ответом будет «0».

3. Определение позиции первой единицы (все нули до нее не подсчитываются)

4. Если единица существует и не находится на последней позиции строки, то начинать подсчет значащих нулей. В противном случае, значащих нулей не существует, ответ «0».

5. Для подсчета значащих нулей вводится и обнуляется переменная-счетчик.

6. Перебирая строку с найденной позиции первой единицы до конца строки, подсчитывается количество нулей

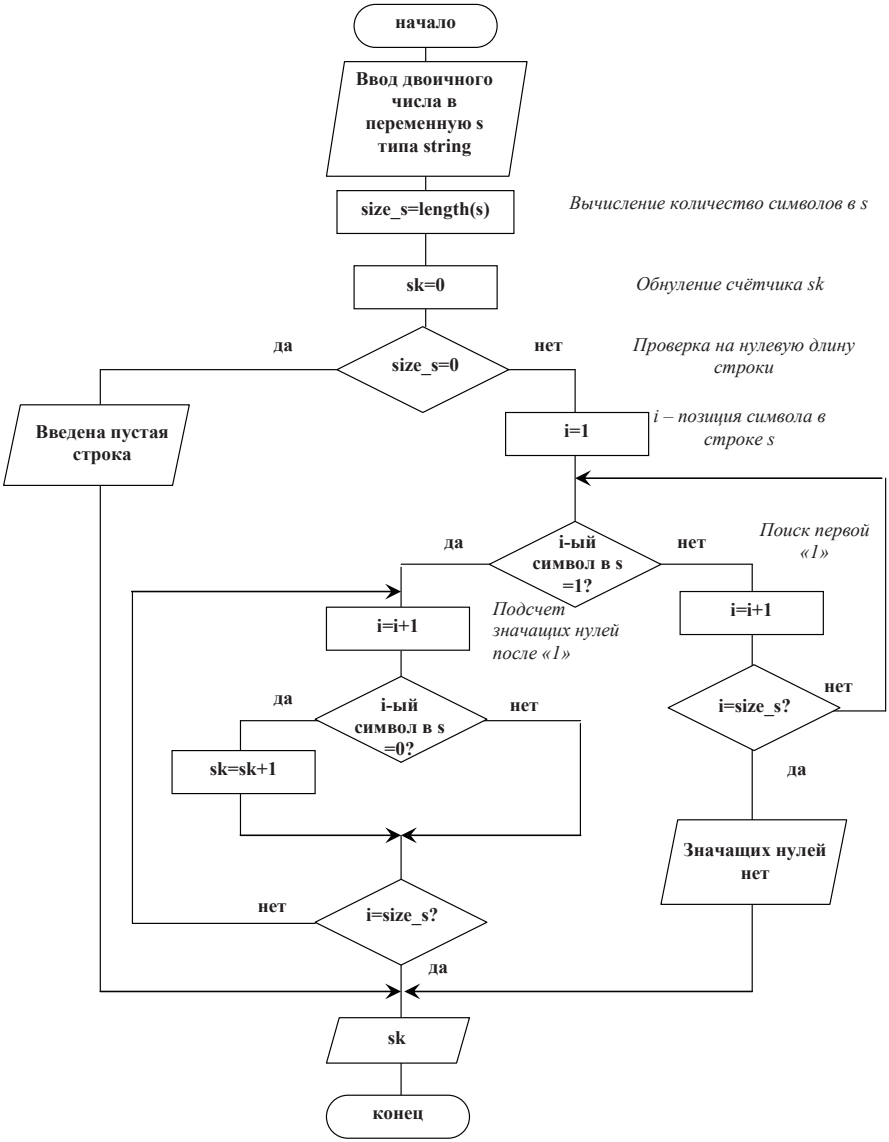
7. Вывод результата

Ниже приведены решения задачи как в виде блок-схемы, реализующей алгоритм поставленной задачи, так и в виде программных кодов (языки программирования VBA, Pascal ABC, C#).

Решение на Pascal ABC

```
label 1;
var s:string;
size_s,sk,i,j:integer;
begin
readln(s);
size_s:=Length(s);
sk:=0;
if (size_s<>0) then
begin
  for i:=1 to size_s do
  if Copy(s,i,1)='1' then goto 1;

1:if i<size_s then
  for j:=i to size_s do
  if Copy(s,j,1)='0' then sk:=sk+1;
end;
writeln(sk);
end.
```

Блок-схема алгоритма задачи



Решение задачи на VBA.

```
Sub task()
Dim s As String
s = InputBox("Введите число")
size_s = Len(s)                              'определение длины строки
sk = 0                                       'обнулении счетчика
If size_s <> 0 Then                          'если введенная строка нулевая, то окончание работы
    For i = 1 To size_s                      'поиск от первой до последней позиции символа"1"
    If Mid(s, i, 1) = "1" Then Exit For      'при нахождении символа "1" - досрочный выход из цикла
    Next i

    If i < size_s Then                       'если "1" не нашлась или оказалась последней, то окончание работы
        For j = i To size_s                  'поиск от позиции первой единицы до конца строки
        If Mid(s, j, 1) = "0" Then sk = sk + 1  'если есть "0", увеличить счетчик
        Next j
    End If
End If
MsgBox (sk)
End Sub
```

По поводу алгоритма, реализованного на Паскале, стоит дать некоторое уточнение: в алгоритме используется оператор goto, который не рекомендуется к активному использованию в программировании. Этому можно привести следующие объяснения:

• Оператор goto позволяет совершать переходы из любой точки программы в любую другую, что при некорректном его использовании может привести к значительным сложностям отладки программы

• Современные средства программирования позволяют писать программы без использования goto.

Так, в VBA, для досрочного выхода из цикла For next используется оператор ExitFor. В Pascal ABC и в C# для аналогичных целей используется break.

Таким образом, последний алгоритм можно переписать как

```
var s:string;
size_s,sk,i,j:integer;
begin
readln(s);
size_s:=Length(s);
sk:=0;
if (size_s<>0) then
begin
  for i:=1 to size_s do
  if Copy(s,i,1)='1' then break;

  if i<size_s then
  for j:=i to size_s do
  if Copy(s,j,1)='0' then sk:=sk+1;
end;
writeln(sk);
end.
```

Оба алгоритма реализуют абсолютно одинаковую логику. Применение оператора goto не усложняет алгоритм и в данном случае его применение не считается ошибкой. Оба решения засчитываются как

верные. Но стоит помнить, что избыточное и необоснованное применение оператора goto может быть воспринято при проверке задачи как демонстрация незнания основных операторов языка.
Другим способом решить ту же задачу является использование операторов языка, автоматически реализующих поставленную задачу: ниже приведен алгоритм, в котором позиция первой единицы определяется оператором `Pos('1',s)` (найти в строке s позицию строки ‘1’). Естественно, демонстрация хорошего знания текстовых (или иных операторов) учитывается при проверке. Но в подобных ситуациях рекомендуется либо детальное знание операторов, либо использования более простого способа решения задачи.

```
var s:string;
size_s,sk,i,j:integer;
begin
readln(s);
size_s:=Length(s);
sk:=0;
if (size_s<>0) then
begin
  i:=Pos('1',s);

  if i<size_s then
  for j:=i to size_s do
  if Copy(s,j,1)='0' then sk:=sk+1;
  end;
writeln(sk);
end.
```

Окно вывода

```
00000
6
```

Как видно из приведенного скриншота, при введенной строке пять нулей «00000», программа в качестве ответа выводит число 6, которое не является верным. Причиной тому является специфика оператора `Pos('1',s)`, который при отсутствии искомой строки (а при указанных введенных данных единица отсутствует) в качестве результата работы дает число «0». Что приводит к дальнейшему некорректному выполнению алгоритма.

В данной ситуации требуется ввести дополнительную проверку:

```
if (i<size_s) and (i>0)
```

```
var s:string;
size_s,sk,i,j:integer;
begin
readln(s);
size_s:=Length(s);
sk:=0;
if (size_s<>0) then
begin
  i:=Pos('1',s);

  if (i<size_s) and (i>0)  then
  for j:=i to size_s do
  if Copy(s,j,1)='0' then sk:=sk+1;
  end;
writeln(sk);
end.
```

Ниже приводится решение задачи на C# двумя способами: первый – описанный в блок-схеме, с поиском первой единицы перебором строки. Второй – с использованием `i=s.IndexOf("1");` , работающим аналогично `Pos('1',s)` в Pascal ABC.

```
static void Main(string[] args)
        {
            string s;
            int  i,j,size_s,sk;
            s = Console.ReadLine();
            size_s = s.Length;
            sk = 0;
            if (size_s != 0)
            {
                for (i = 0; i < size_s; i++)
                    if (s.Substring(i,1)=="1") break;

                for (j = i; j < size_s; j++)
                    if (s.Substring(j, 1) == "0") sk++;
            }
            Console.WriteLine(sk);
        }

static void Main(string[] args)
        {
            string s;
            int  i,j,size_s,sk;
            s = Console.ReadLine();
            size_s = s.Length;
```

```
        sk = 0;
        if (size_s != 0)
        {
            i=s.IndexOf("1");

            for (j = i; j < size_s; j++)
                if (s.Substring(j, 1) == "0") sk++;

        }
        Console.WriteLine(sk);
    }
```

Таким образом, на детальном разборе данной задачи продемонстрировано, что решения задач могут быть оформлены различными способами, при реализации программного кода может быть выбран любой существующий язык программирования, алгоритмы могут различаться. Но обязательно

• должны учитываться система счисления вводимого в программу числа

• тип переменных, содержащих введенные значения

• должны отсутствовать математические и логические ошибки

• рекомендуется демонстрация четкого алгоритмического мышления.

Типичные ошибки, допущенные при реализации задачи

1. Неверное определение типов данных и вытекающие ошибки обработки данных

2. Пропуск пункта 3 алгоритма «Определение позиции первой единицы» и подсчет всех нулей в записи числа

3. Пропуск пункта 2 алгоритма «Анализ длины строки: если строка нулевая, то дальнейшее выполнение программы не требуется» не является существенной ошибкой, но наличие в алгоритме данного пункта означает, что автор программы умеет анализировать вводимые значения и оптимизировать алгоритм. Аналогичная ситуация с пунктом 4 алгоритма «Если единица существует и не находится на последней позиции строки».

Задача 5. (Заключительный этап олимпиады школьников «Ломоносов-2015» по информатике (5–9 классы))

Переведите число 2211201122110201 из троичной системы счисления в систему счисления с основанием 27. В качестве цифр используйте десятичные цифры и заглавные латинские буквы.

Проверяется

Понимание сути любой позиционной системы счисления.

Обсуждается

Соответствие между системами счисления в случае, если основание одной системы счисления является степенью другой

Теоретическое сопровождение темы. Решение

Для быстрого и простого решения данной задачи достаточно увидеть, что 27 (основание новой системы счисления) является третьей степенью тройки (основание текущей системы счисления представления числа): $3^3=27$. Поэтому, аналогично предыдущей (четвертой) задаче, преобразование числа из одной системы счисления в другую можно выполнить без арифметических расчетов: каждые три разряда исходного числа троичной системы счисления кодируются одним разрядом числа 27-ричной системы счисления в соответствии со следующей таблицей.

Запись числа в троичной системе счисления			Запись числа в 27-ричной системе счисления	Запись числа в 10-тичной системе счисления
3-й разряд	2-й разряд	1-й разряд		
0	0	0	0	0
0	0	1	1	1
0	0	2	2	2
0	1	0	3	3
0	1	1	4	4
0	1	2	5	5
0	2	0	6	6
0	2	1	7	7
0	2	2	8	8
1	0	0	9	9
1	0	1	A	10

Запись числа в троичной системе счисления			Запись числа в 27-ричной системе счисления	Запись числа в 10-тичной системе счисления
3-й разряд	2-й разряд	1-й разряд		
1	0	2	B	11
1	1	0	C	12
1	1	1	D	13
1	1	2	E	14
1	2	0	F	15
1	2	1	G	16
1	2	2	H	17
2	0	0	I	18
2	0	1	J	19
2	0	2	K	20
2	1	0	L	21
2	1	1	M	22
2	1	2	N	23
2	2	0	O	24
2	2	1	P	25
2	2	2	Q	26

Запишем исходное число 2211201122110201, разбив по три разряда, начиная с младшего:

002 211 201 122 110 201

Теперь каждую группу символов перекодируем одним символом 27-миричной системы счисления (в таблице эти группы выделены цветом):

2 M J H C J

Это и будет ответ $2MJHCJ_{27}$

Типичные ошибки, допущенные при реализации задачи

1. Математические ошибки
2. Преобразование заданного числа с десятичную систему счисления с последующим переводом в 27-миричную: задача может быть решена правильно, но с потерей большого количества времени.

Задача 6 (11 класс, Роботрон, отборочный тур 2016 год)

Исполнитель Черепашка перемещается на экране компьютера, оставляя след в виде линии. В каждый конкретный момент известно

положение исполнителя и направление его движения. У исполнителя существуют две команды:

Вперед x (x – число) – вызывает передвижение Черепашки на x шагов в направлении движения.

Направо m (m – число) – вызывает изменение направления движения на m градусов по часовой стрелке.

Запись Повтори k [Команда 1 Команда 2 … Команда q] означает, что последовательность q команд в скобках повторится k раз.

Напишите программу для данного исполнителя, которая приведет к появлению на экране правильного n-угольника со стороной A за минимальное число шагов.

Укажите, какие параметры программы возможно изменить с сохранением исходного результата выполнения программы. Какие допустимые значения изменяемых параметров? Как их изменение повлияет на выполнение программы?

Проверяется

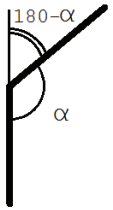
При решении данной задачи требуется продемонстрировать понимание команд как компонента программы; понимание параметров команд; понимание области допустимых значений параметров и геометрическую подготовку.

Обсуждается

Время выполнения задачи; оптимальность алгоритма. Допустимые значения параметров.

Геометрическое решение поставленной задачи; различные варианты; оценивается удачность вариантов с позиции реализации каждого варианты программным кодом.

Теоретическое сопровождение темы. Решение



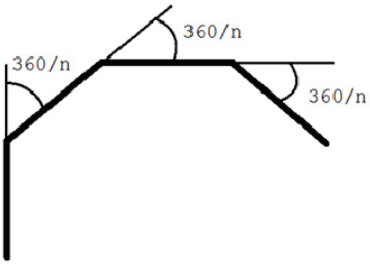
Из курса геометрии известно, что сумма углов правильного n-угольника равна 180*(n-2).

Следовательно, каждый угол правильного n-угольника α равен 180*(n-2)/n.

Чтобы нарисовать правильный n-угольник надо из исходной точки продвигаться вперед на длину стороны A, а далее – делать поворот на угол 180-α

$$180 - \alpha = 180 - 180 * \frac{n-2}{2} = 180 * \left(1 - \frac{n-2}{n}\right) = 180 * \frac{n-n+2}{n} = 180 * \frac{2}{n} = \frac{360}{n}$$

Таким образом, для рисования правильного n-угольника надо выполнять следующую цепочку действий:



Вперед A
Направо 360/n
Вперед A
Направо 360/n
…
Вперед A
Направо 360/n
Вперед A

} n-1 раз

Используя команду цикла, получаем конечный результат

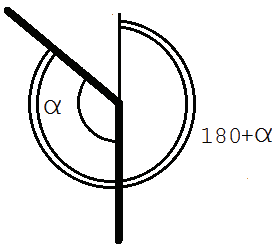
Повтори n-1 [Вперед A Направо 360/n]
Вперед A

Или полностью ему аналогичный по количеству выполняемых команд

Вперед A
Повтори n-1 [Направо 360/n Вперед A]

Оба ответа равносильны и верны.
Вариант ответа

Повтори n [Вперед A Направо 360/n]



выглядит более лаконичным, но, по мере выполнения программы, выполняет одну команду (последний поворот) лишнюю.

Альтернативным вариантом решения является ситуация, приведенная на рисунке: многоугольник рисуется против часовой стрелки.

В этом случае угол поворота определяется как 180+α

$$180 + \alpha = 180 + 180 * \frac{n-2}{2} = 180 * \left(1 + \frac{n-2}{n}\right) = 180 * \frac{n+n-2}{n} =$$

$$= 180 * \frac{2*n-2}{n} = 180 * \frac{2*(n-1)}{n} = 360 * \frac{n-1}{n} = 360 * \left(1 - \frac{1}{n}\right)$$

Саму программу можно записать как

Повтори n-1 [Вперед А Направо 360*(n-1)/n]
Вперед А

Данный вариант, хотя и не содержит математических ошибок, является менее удачным с позиции программирования: при сравнении времени выполнения команды

Направо 360/n

и времени выполнения команды

Направо 360*(n-1)/n

получается, что первая команда будет выполняться быстрее за счет меньшего количества математических действий при расчете значения параметра. Следовательно, данный вариант не является оптимальным.

В задаче также требуется определить допустимые значения параметров программы и определить, как их изменение повлияет на выполнение программы.

Параметр А – определяет сторону многоугольника и, как следствие, конечный размер фигуры. Допустимое значение параметра А – любое число:

• дробное значение возможно;

• при отрицательном значении параметра фигура будет прорисовываться в обратную сторону

• нулевое значение параметра приводит к рисованию точки

Изменение значения параметра А в команде

Вперед А

Неизбежно изменит конечный результат

Параметр n определяет количество углов многоугольника. Количество углов не может быть ни дробным, ни отрицательным. Из геометрических соображений n должно быть больше 2. Получается, что допустимое значение параметра n - целое число больше 2.

Аналогично параметру А изменение значения параметра n в команде

Направо 360/n

неизбежно изменит конечный результат.

Но увеличение значения параметра n-1 в команде

Повтори n-1

не изменит рисуемую фигуру. Вместо n-1 можно указать любое целое число, большее, чем n-1. Естественно, увеличится время выполнения программы.

Решения могут содержать иные, математически непротиворечащие приведенному ответу, варианты ответов. Но обязательно

• должны отсутствовать математические и логические ошибки

• учитывается время выполнения предлагаемой программы (согласно условию предполагается минимальное число шагов)

• учитывается понимание влияния параметров на выполнение программы

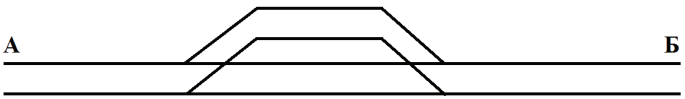
• учитывается понимание допустимых значений параметров.

Типичные ошибки, допущенные при реализации задачи

1. Ошибки в математических расчетах
2. Неоптимальное число шагов выполнения программы
3. Неправильное определение области допустимых значений параметров
4. Решение задачи в частном виде (например, для треугольника, четырехугольника и пр.)

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 верёвок 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),

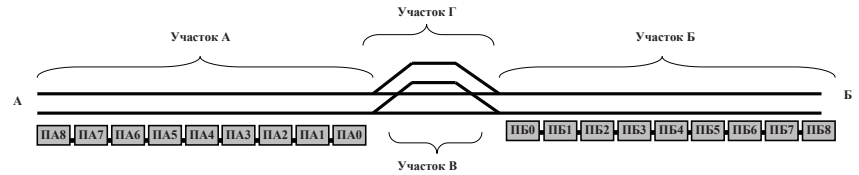
– при этом ПА0 - локомотив,

– а ПА1 - ПА8 - вагоны первого поезда с первого по восьмой.

• ПБ – обозначение второго поезда (который двигался со стороны Б).

– при этом ПБ0 - локомотив,

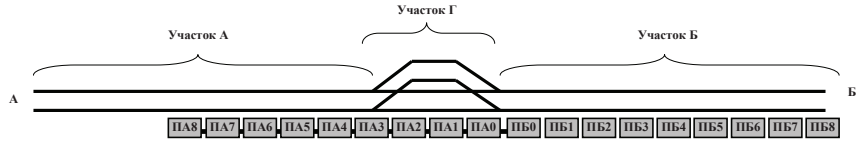
– а ПБ1 - ПБ8 - вагоны второго поезда с первого по восьмой.



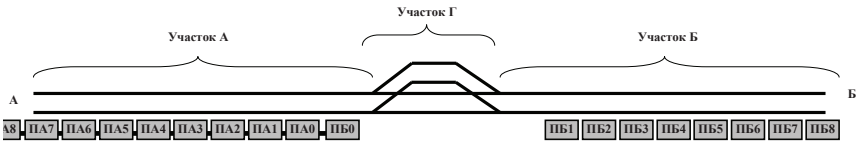
Общая идея алгоритма следующая: вагон поезда ПБ загоняется в боковое ответвления, затем поезд ПА проезжает в Б. Прицепляет загнанный вагон к своему хвосту и перегоняет его в А. Таким образом, по одному вагону составы меняются местами.

Детальное описание алгоритма:

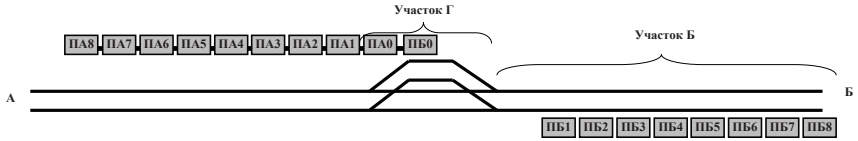
расцепить поезд ПБ
ПА вперед в Б
к ПА прицепить ПБ0



ПА назад в А

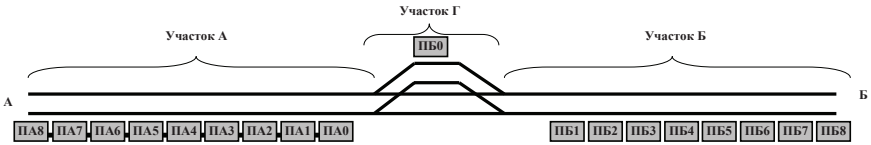


ПА вперед в Г

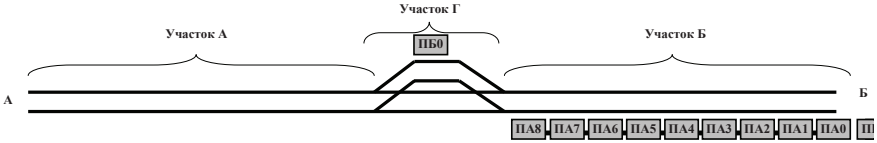


отцепить ПБ0 в Г

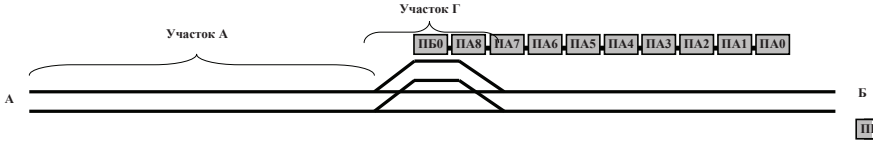
ПА назад в А



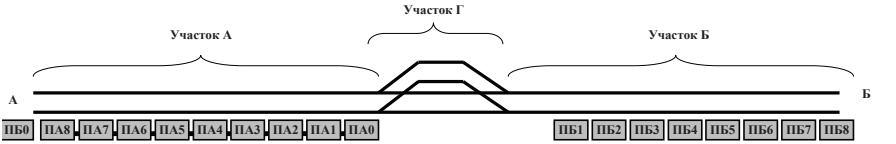
ПА вперед в Б



ПА назад в Г
к ПА прицепить ПБ0

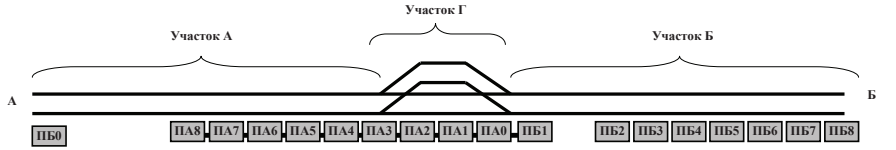


ПА назад в А
отцепить ПБк в А



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

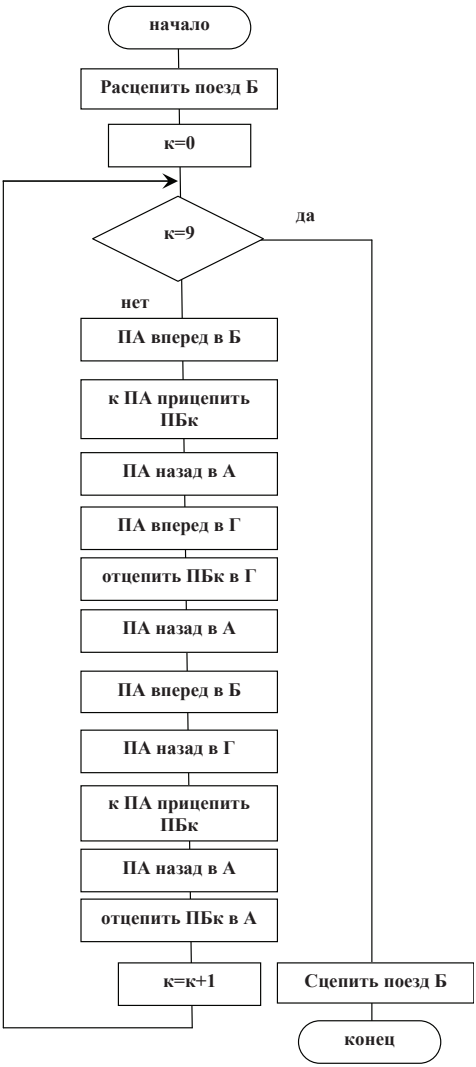
Последним действием будет сцепить поезд ПБ.

Очевидно, что действия по переброске 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 к=0 до к=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 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 и дается пример использования подпрограмм (в частности – функции).

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

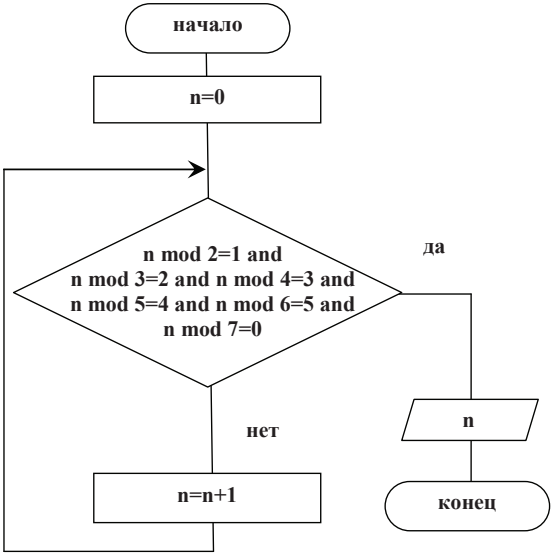
Теоретическое сопровождение темы. Решение

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 n, в то время, как простейших арифметический знаний достаточно, чтобы увидеть значения n, которые не подходят. Соответственно, нет смысла увеличивать количество проверяемых значений.

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




```
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.
```

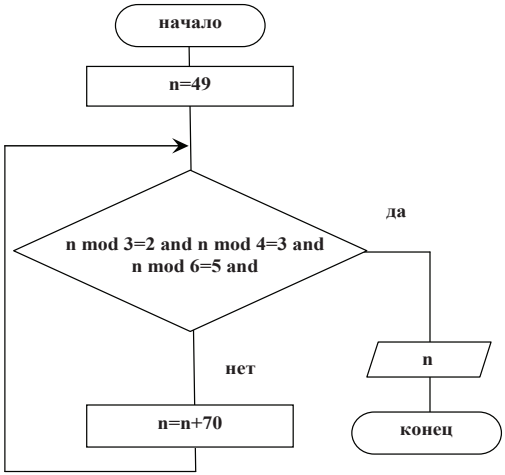
Начнем с первого недостатка: если на 7 это число делится нацело, значит, данное число является множителем семерки. Таким образом, в алгоритме команду n=n+1 можно изменить на n=n+7. Если число, будучи разделено на 2, дает в остатке 1, следовательно, число нечетное; все четные множители семерки исключаются.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

Если сделать дальнейшие математические наблюдения, то из факта, что при делении на 5, число дает в остатке 4, можно сделать вывод, что число заканчивается либо на 4, либо на 9. Четным оно быть не может, поэтому, число должно заканчиваться на 9.

Знание таблицы умножения подсказывает, что наименьшее нечетное число, делящееся на 7 и заканчивающее на 9 – это 49; следующее число – это 49+70=119; следующее 119+70=189, и т.д.

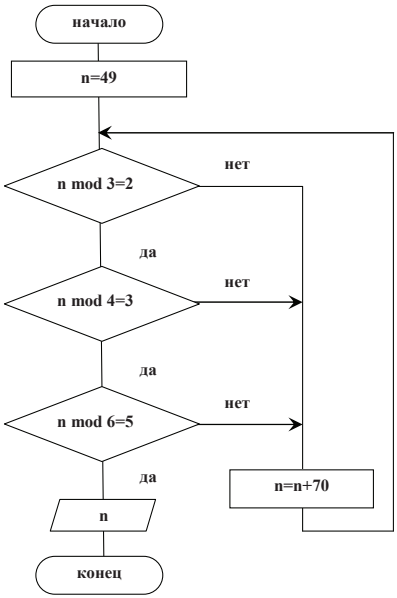
Измененный алгоритм представлен ниже. Обратите внимание, что в алгоритме за счет изменение начального значения n и шага изменения убраны проверки деления на 2, на 5 и на 7, что также увеличивает скорость выполнения программы.




```
var n:integer;
begin
n:=49;
while not(and (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.

Для этого 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 введение множественных, обсужденных 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 верного 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 однотипными проверками является удачным решением.

Рассмотрим подробнее второй пункт: если какое-то однотипное 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 успешны – 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 (а не досрочном, через break) значение переменной цикла будет (для текущей задачи) равно 7. Ниже приводится вариант кода задачи для различных языков программирования.

Pascal ABC

```
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.
```

C#

```
static                void
Main(string[] args)
{
     int i;
     int n = -21;
     bool b=true;
     while (b)
     {
     n=n+70;
for (i = 3; i < 7; i++)
 if (n % i != i - 1)
break;
if (i==7) b=false;
      }
Console.WriteLine(n);
}
```

VBA

```
Sub task()
n = -21
b = True
While b
    n = n + 70
    For i = 3
To 6
If n Mod i <>
i - 1 Then Ex-
it For
    Next i
If i = 7 Then
b = False
Wend
MsgBox (n)
End Sub
```

Второй недостаток не зависит от языка программирования.
Но, 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 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 ABC

```
var n,i,sk,k:integer;                  ‘описание переменных
b:boolean;                             ‘
begin                                  ‘запрос k
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             ‘печать результата
else sk:=sk+1;
if sk=k-2 then b:=false;
end;
writeln(n);
end.
```

C#

```
static void Main(string[] args)        //
    {                                  //описание и инициализация
int i;                                 //переменных
int n = 0;                             //
bool b=true;                           //запрос k и преобразование в
int k = int.Parse(Console.ReadLine()); // число
    while (b)                          //
    {                                  //
    n=n+k;                             //
    for (i = 3; i < k; i++)            //
    if (n % i != i - 1) break;         //
    if (i==k) b=false;                 //печать результата
    }
Console.WriteLine(n);
    }
```

VBA

```
Sub task()                             ‘описание ‘переменных
k = CInt(InputBox("Введите число"))    ‘запрос 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                    'печать результата
Wend
MsgBox (n)
End Sub
```

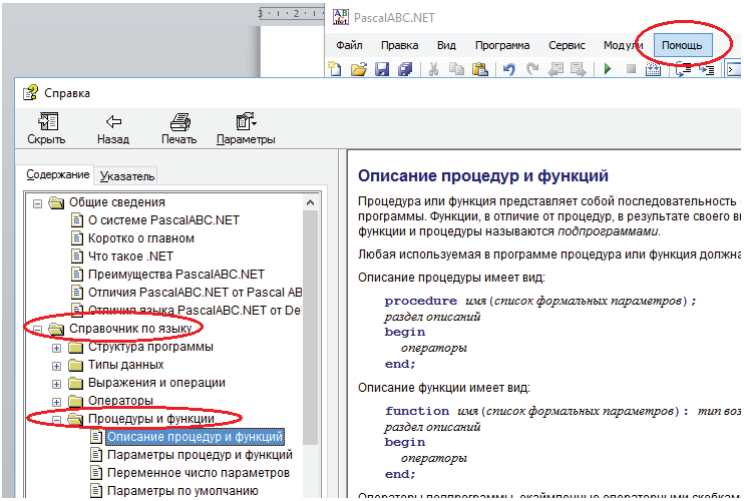
Примечание: если детально проанализировать полученные коды, можно увидеть, что не при любых значениях переменной k программа будет благополучно завершена. Вопрос допустимых значений переменной k предлагаем, как дополнительное задание.

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, как, например, показано 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 случаях 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 входных параметров (указываемых в скобках после имени функции), и возвращает, как результат своей работы, единственное значение.

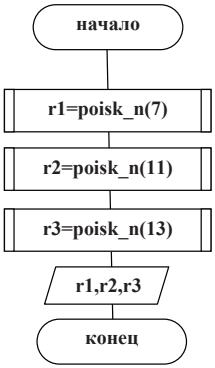
Итак, ниже приведен код функции poisk_n, которая получает на вход значение k, и в качестве результата возвращает найденное значение n. Эта функция будет вызываться из основной программы несколько раз с разными значениями k. Код основной программы сводится к вызову функции с разными параметрами.

Pascal ABC

```
function poisk_n(k:integer):integer;
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
```

Блок-схема алгоритма

```
  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.
```



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. Рекуррентная функция – 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 зацикливанию программы. Поэтому, как правило, в теле функции стоит условие, по выполнению или невыполнению 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*…*n.

Pascal ABC	C#	VBA

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.
	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;
	}
	}
		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

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*…*n можно вычислить как n!=n*(n-1)*(n-2)*…*1: от перестановки мест множителей произведение не меняется. То есть получается, что

n!=n*(n-1)*(n-2)*…*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 работа функции завершается. Тип возвращаемого значения – булевский. Основная программа увеличивает значение n до тех пор, пока функция не возвратит значение False.

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)
```

C#

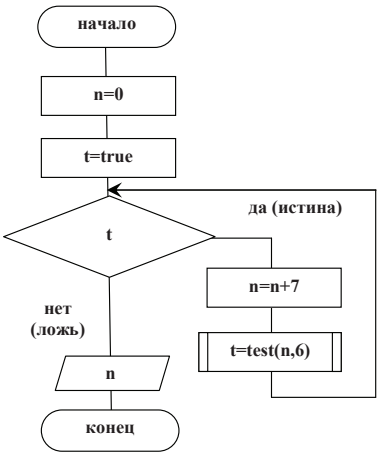
```
class Program
    {
        static void Main(string[] args)
        {
            int n = 0;
            bool b=true;
            while (b)
```

```
    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.
```

```
            {
            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;
            }
    }
```

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
```



Типичные ошибки, допущенные при реализации задачи

1. Математические просчеты
2. Неверное понимание оптимального алгоритма.

Задача 9 (Всероссийская олимпиада школьников по информатике, 2014/15 уч. год Первый (школьный) этап, г. Москва). Задания для 9–11 классов

«Автомобильные номера». В Российской Федерации на разных видах транспортных средств устанавливаются разные по формату регистрационные знаки («автомобильные номера»). Вот пример нескольких возможных форматов регистрационных знаков.

№	Пример	Описание формата	Тип транспортного средства
1	Y01 9KM	Буква, три цифры, две буквы	Частные транспортные средства
2	AB179	Две буквы, три цифры	Общественный транспорт и такси
3	OH2645	Две буквы, четыре цифры	Прицепы
4	3384CT	Четыре цифры, две буквы	Мотоциклы

В этой задаче «буквой» может быть любая заглавная буква латинского алфавита. Напишите программу, которая по регистрационному знаку определяет его тип или определяет, что регистрационный знак некорректен. Программа получает на вход три строки текста, каждая строка содержит один образец регистрационного знака (возможно, некорректный). Каждый образец содержит от 1 до 10 символов, являющихся цифрами и заглавными латинскими буквами (других символов во входных данных быть не может).

Программа должна вывести для каждого образца число, соответствующее типу транспортного средства, как в приведенной таблице, то есть 1 – для частных транспортных средств, 2 – для общественного транспорта, 3 – для прицепов, 4 – для мотоциклов. Если номерной знак некорректен (не подходит ни к одному из указанных типов), то необходимо вывести число 0.

Каждое число необходимо выводить в отдельной строке. Пример входных и выходных данных

Ввод	Вывод
Y019KM	1
A9999	0
OH2645	3

Проверяется

Умение создавать и отлаживать коды программ на каком-либо языке программирования; навыки алгоритмизации, умение работать с символьными и с текстовыми типами данных.

Естественно, для выполнения данной программы требуется наличие аппаратных и программных средств. Независимо от условий проводимой олимпиады (проводится она в дисплейном классе или в письменном виде) представляется полезным разобрать программную реализацию данной задачи.

Обсуждается

Символьные и строковые типы данных.
Применение функций.
Оптимальность алгоритма.

Теоретическое сопровождение темы. Решение

Алгоритм решения данной задачи не сложен: требуется ввести номер транспортного средства и проанализировать введенный набор символов на соответствие существующим форматам. Фактически задача сводится к посимвольному сравнению заданного номера. Для ее успешного решения надо понимать, что такое символьный тип данных. Частично специфика работы со строковыми типами данных была разобрана при решении задачи №4.

Переменная типа string представляет собой последовательность отдельных символов. Причем, переменная типа string может не содержать ни одного символа. В Pascal ABC и в C# строка – это массив символов. Для строк в любых языках программирования введены операции сложения, извлечения части подстроки (слева, справа, из любой позиции), определения позиции искомой подстроки в строке, определения длины строки.

Переменная типа char содержит только один символ. Преимущество символьных данных заключается в том, что для символьных переменных доступны операции получения соответствующего символу UNICODE и операция обратная ей – по UNICODE вернуть символ.

На приведенном рисунке видно, что UNICODE символа ‘s’ равен 115, а UNICODE следующего за ним символа ‘t’ равен 116. Таким образом, для символьных переменных есть операция сравнения. Сравнение производится по UNICODE. Получается, что ‘s’<’t’.

A ‘S’<’s’.

```
begin
ch:='s';
writeln(ordUnicode(ch));
ch:='t';
writeln(ordUnicode(ch));
ch:='S';
writeln(ordUnicode(ch));
end.
```

Окно вывода

```
115
116
83
```

Именно это свойство символьных переменных (сравнение символов по UNICODE) и применяют для проверки соответствия какого-либо символа заданной группе символов:

• Если (согласно условию задачи) надо проверить, не является ли символ цифрой, то данный символ должен быть больше или равен ‘0’ и одновременно меньше или равен ‘9’.

• Если символ проверяется на принадлежность к латинскому алфавиту (согласно условию задачи – только заглавные буквы), то он должен быть больше или равен ‘A’ и меньше или равен ‘Z’.

Вывод: для проведения сравнения введенного номера с существующими шаблонами надо будет сравнивать каждый символ введенного номера. Но, согласно условию, вводится номер целиком (не по одному символу), то есть строка. Для разрешения данного вопроса в Pascal ABC и в C# надо вспомнить, что строка – это массив символов. На рисунке ниже приведена программа работы со строковой переменной s.

```
var s:string;
begin
s:='abcdefg';           //в переменнную s записывается строка 'abcdefg'
writeln(s[2]);          //на экран выводится второй элемент строки
writeln(s[5]);          //на экран выводится пятый элемент строки
writeln(s[s.Length]);   //на экран выводится последний элемент строки
end.
```

Окно вывода

```
b
e
g
```

Из приведенного рисунка видно, что s, объявленная как строковая переменная позволяет обращаться к себе, как к массиву:

• s[2] – обращение ко второму элементу массива s, то есть символ ‘b’;

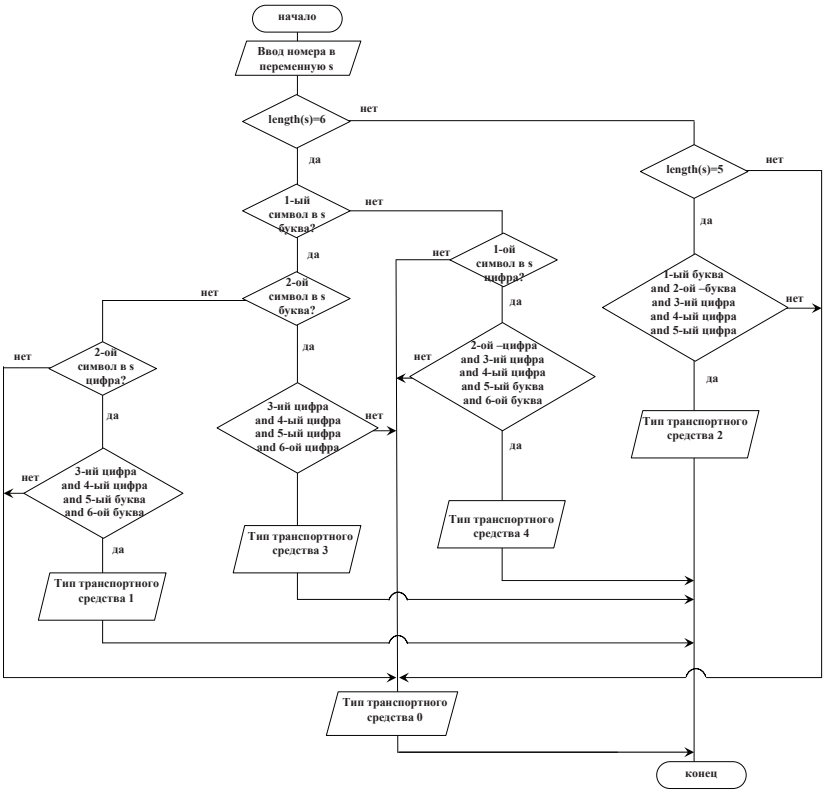
• s[5] – обращение к пятому элементу массива s, то есть символ ‘e’.

• s[s.Length] – обращение к элементу массива s, с номером s.Length. s.Length – это определение длины строки s. Следовательно, s[s.Length] – символ ‘g’.

В VBA каждый символ строки извлекается при помощи команды Mid(s, 1), которая позволяет извлечь подстроку из строки s длиной (в данном случае) 1 символ.

Сравнивая символы введенного номера важно помнить, что введенный номер – строка, которая может содержать от одного до десяти символов. А существующие форматы описывают номера в 5 и 6 символов. Таким образом, представляется целесообразным предварительно проверить длину введенной строки, не проводя дальнейших проверок, если длина не соответствует 5 или 6.

Таким образом, определив способы сравнения номера с шаблонами, можно представить алгоритм решения задачи в виде блок-схемы.



Анализ блок-схемы показывает, что

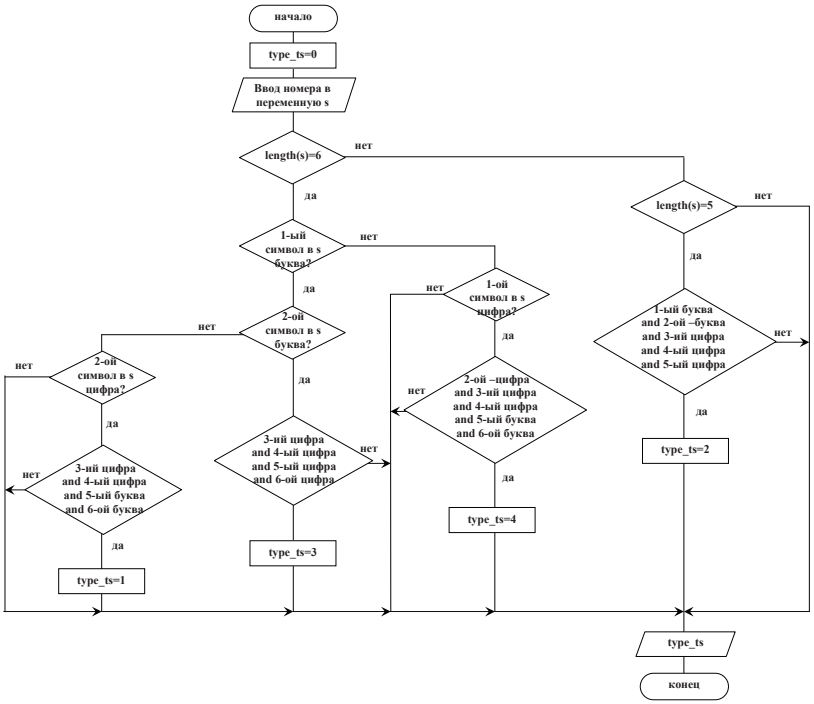
• Практическая реализация задачи будет строиться на слишком большом количестве последовательно примененных операторов условного перехода (обратите внимание, что каждая проверка каждого символа – это также двойное условие).

• Логика приведенного алгоритма затруднена и грозит применением непопулярного оператора goto для реализации выхода из задачи по несовпадению ни с одним из указанных форматов.

Для исправления указанных недостатков

• Рекомендуется ввести функции булевского типа для проверки, является ли символ цифрой (test_num) или заглавной буквой (test_char) латинского алфавита. Обе функции принимают символ в качестве параметра; анализируют его на принадлежность требуемому диапазону и возвращают true, если принадлежит; false, если нет.

• Ввести переменную type_ts (тип транспортного средства) и определить ее значение равным нулю до начала работы алгоритма. Только в случае совпадения с одним из указанных форматов значение переменной будет переопределено.



Далее приводятся коды программ, реализующих данный алгоритм. Естественно, можно привести и иные, более удачные варианты решения данной задачи. При выборе алгоритма авторы руководствовались

• соображениями наибольшей очевидности алгоритма и программного кода;

• возможностью однотипной реализации алгоритма на языках PascalABC, C#, VBA.

Некоторым отклонением от исходной задачи является организация ввода номеров до тех пор, пока не будет введена пустая строка (нулевой длины), что позволяет проверить все ветви алгоритма однократным запуском программы. Для этого применен оператор цикла по условию while с условием продолжения цикла ненулевой длиной введенной строки.

Для приведения приведенных кодов в полное соответствие с исходной задачей достаточно заменить цикл по условию на цикл по счетчику с тремя повторениями.

При анализе кодов программ обратите внимание, что имеются частичные несовпадения кодов, что связано со спецификой каждого языка:

• В C# нумерация любого массива начинается с нуля. Строка – это массив символов, следовательно, первым символом является s[0].

• В Pascal ABC строка – это также массив символов. Но нумерация символов в массиве строки начинается с 1, следовательно, первым символом является s[1]. Ввод данных в программе заканчивается вводом символа ‘0’.

• В VBA не используется тип char, но есть возможность производить сравнение строк согласно UNICODE их символов: «большей» является та строка, в которой UNICODE соответствующего символа выше. Кроме того строка не представляется массивом символов, поэтому в процедуру передается не символ, а подстрока, выделенная из проверяемой строки.

C#:

```
class Program
    {
        static bool test_char(char tc)
        {
            if ((tc >= 'A') && (tc <= 'Z')) return true;
            else return false;
        }
```

```
        static bool test_num(char tc)
        {
            if ((tc >= '0') && (tc <= '9')) return true;
            else return false;
        }
        static void Main(string[] args)
        {
            string s = Console.ReadLine();
            while (s.Length != 0)
            {
                int type_ts = 0;
                if (s.Length == 6)
                {
                    if (test_char(s[0]))
                    {
                        if (test_char(s[1]))
                            if (test_num(s[2]) && test_num(s[3])
&& test_num(s[4]) && test_num(s[5])) type_ts = 3;
                        if (test_num(s[1]))
                            if (test_num(s[2]) && test_num(s[3])
&& test_char(s[4]) && test_char(s[5])) type_ts = 1;
                    }
                    else
                    if (test_num(s[0]))
                        if (test_num(s[1]) && test_num(s[2]) &&
test_num(s[3]) && test_char(s[4]) && test_char(s[5])) type_ts = 4;
                }
                else
                {
                    if (s.Length == 5)
                    {
                        if (test_char(s[0]) && test_char(s[1]) &&
test_num(s[2]) && test_num(s[3]) && test_num(s[4])) type_ts = 2;
                    }
                }
                Console.WriteLine("тип транспортного средства
{0}", type_ts);
                s = Console.ReadLine();
            }
        }
    }
```

Pascal ABC:

```
function test_char(tc:char):boolean;
begin
if (tc>='A') and (tc<='Z') then test_char:=true else
test_char:=false;
end;
```

```
function test_num(tc:char):boolean;
begin
if   (tc>='0')   and   (tc<='9')   then   test_num:=true   else
test_num:=false;
end;
var s:string;
type_ts:integer;
begin
readln(s);
  while s<>'0'   do
  begin
  type_ts:=0;
  if (s.Length=6) then
  begin
    if test_char(s[1]) then
    begin
      if test_char(s[2]) then
        if
(test_num(s[3]))and(test_num(s[4]))and(test_num(s[5]))and(te
st_num(s[6])) then type_ts:=3;
      if test_num(s[2]) then
        if
(test_num(s[3]))and(test_num(s[4]))and(test_char(s[5]))and(t
est_char(s[6])) then type_ts:=1;
    end
    else
    if test_num(s[1]) then
      if
(test_num(s[2]))and(test_num(s[3]))and(test_num(s[4]))and(te
st_char(s[5]))and(test_char(s[6]))then type_ts:=4;
  end
  else
    if (s.Length=5) then
      if
(test_char(s[1]))and(test_char(s[2]))and(test_num(s[3]))and(
test_num(s[4]))and(test_num(s[5])) then type_ts:=2;
  writeln(type_ts);
  readln(s);
  end;
end.
```

VBA:

```
Function test_char(tc As String)
If tc >= "A" And tc <= "Z" Then
test_char = True
Else
test_char = False
End If
End Function
Function test_num(tc As String)
```

```
If tc >= "0" And tc <= "9" Then
test_num = True
Else
test_num = False
End If
End Function
Sub task()
Dim s As String
Dim type_ts As Integer
s = InputBox("Введите номер")
While Len(s) <> 0
type_ts = 0
    If Len(s) = 6 Then
        If test_char(Mid(s, 1)) Then
            If test_char(Mid(s, 2)) Then
                If test_num(Mid(s, 3)) And _
                test_num(Mid(s, 4)) And _
                test_num(Mid(s, 5)) And _
                test_num(Mid(s, 6)) Then type_ts = 3
            Else
                If test_num(Mid(s, 2)) Then
                    If test_num(Mid(s, 3)) And _
                    test_num(Mid(s, 4)) And _
                    test_char(Mid(s, 5)) And _
                    test_char(Mid(s, 6)) Then type_ts = 1
                End If
            End If
        End If
        If test_num(Mid(s, 1)) Then
            If test_num(Mid(s, 2)) And _
            test_num(Mid(s, 3)) And _
            test_num(Mid(s, 4)) And _
            test_char(Mid(s, 5)) And _
            test_char(Mid(s, 6)) Then type_ts = 4
        End If
    Else
        If Len(s) = 5 Then
            If test_char(Mid(s, 1)) And _
            test_char(Mid(s, 2)) And _
            test_num(Mid(s, 3)) And _
            test_num(Mid(s, 4)) And _
            test_num(Mid(s, 5)) Then type_ts = 2
        End If
    End If
MsgBox (type_ts)
s = InputBox("Введите номер")
Wend
End Sub
```

Типичные ошибки, допущенные при реализации задачи

1. Отсутствие функций проверки символа на число или на заглавную латинскую букву позволяет реализовать данный алгоритм, но значительно его затрудняет. В подобных случаях рекомендуется применение функций проверки.

2. Ошибки реализации логики алгоритма, как правило – в применения оператора if.

3. Избыточные проверки уже проверенных символов. Например, после успешной проверки первого символа на число остается проверить оставшиеся 3 символа на число и 2 символа на букву латинского алфавита. Указанный недочет не является фатальной ошибкой, но увеличивает время выполнения программы и не лучшим образом характеризует логические навыки автора программы.

4. Применение goto для выхода в случае неудачного прохождения проверки. В данной задаче наиболее удобным вариантов является исходное определения значения типа транспортного средства равным нулю. В случае удачного прохождения проверки значение типа переопределяется в соответствии со стандартом.

Задача 10 (Заочный тур III Открытой олимпиады школьников по программированию НИТУ«МИСиС» и Cognitive Technologies)

«Сложение-вычитание»

Ограничение времени: 1 секунда

Ограничение памяти: 64 Мегабайта

Ввод: Стандартный поток ввода (stdin)

Вывод: Стандартный поток вывода (stdout)

Коля только что научился складывать и вычитать многозначные десятичные числа. Это ему так понравилось, что он все свободное время отдает придуманной им игре. Он выписывает строчку десятичных цифр, вписывает между некоторыми парами соседних цифр знаки “+” и “-” и подсчитывает результат. Например, строчку 51349 он может превратить в 51+3-49, что дает в результате число 5. Однако Коля не умеет так расставить знаки, чтобы получить заданное число n. Можете ли вы помочь Коле?

Знаки “ + ” и “ - ” нельзя ставить в начале и в конце последовательности, только между некоторыми соседними цифрами. Некоторые числа в получившемся выражении могут начинаться с нулей (например, если изначальная строка задана как 5007, а результат

должен быть 57, то знаки можно расставить следующим образом: 50 + 07 = 57).

Исходные данные

В первой строке записана последовательность из k десятичных цифр, первая из которых отлична от нуля ($1 \le k \le 12$).

В следующей строке записано целое число n ($0 < n < 10^{18}$).

Результат

Выведите единственную строку, как показано в примерах. Если возможных результатов несколько, выведите тот, в котором число вставленных знаков минимально. Если же и таких несколько, выведите любое. Если расставить знаки требуемым образом невозможно, выведите слово «impossible» (без кавычек).

Пример:

Исходные данные	Результат
51349 5	51+3-49=5
51349 500	513-4-9=500
51349 100	impossible

Проверяется

Умение создавать и отлаживать коды программ на каком-либо языке программирования; навыки алгоритмизации, умение работать с текстовыми и числовыми данными. Знание типов данных, особенностей работы с различными типами данных, преобразование типов данных.

Аналогично предыдущей задаче, для выполнения данной программы требуется наличие аппаратных и программных средств. Независимо от условий проводимой олимпиады также представляется полезным разобрать программную реализацию данной задачи.

Обсуждается

Массивы. Преобразование типов данных. Ввод данных из файла. Оптимальность алгоритма.

Теоретическое сопровождение темы. Решение

Основная идея: для того, чтобы найти решение (одно или все возможные) надо перебрать все варианты расстановок символов «+» и

«-» между k символами введенной строки. Для каждого варианта расстановки рассчитать результат расстановки и сравнить его с числом n; при совпадении результата вычисления и числа n, рассмотреть данную расстановку как вариант ответа. Из вариантов ответа в качестве лучшего надо выбрать тот, в котором минимальное количество символов «+» и «-».

Алгоритм можно представить как последовательность следующих этапов:

• Ввод данных

• Формирование вариантов расстановок

• Подсчет суммы по каждому варианту.

• Сравнение суммы с числом n. При совпадении – поиск наилучшего результата.

• Печать результата; при отсутствии вариантов – вывод слова “impossible”.

При практической реализации данной задачи возникает ряд сложностей. Разберем последовательно реализацию каждого этапа алгоритма.

0 этап: ввод данных

Для простоты объяснения задачи воспользуемся стандартным, консольным вводом/выводом данных – с клавиатуры, через окно ввода/вывода:

Pascal ABC	C#	VBA
`readln(s);` `readln(n);`	`string s = Console.ReadLine();` `int n=int.Parse(Console.ReadLine());`	`s = InputBox("Введите число")` `n= InputBox("Введите результат")`

Но в приведенной задаче входные данные поступают из стандартного потока. Поэтому, в рамках демонстрации альтернативных способов ввода исходной информации, не производя детальный разбор данной темы, приведем код считывания исходных данных их потока.

Поток – это абстракция последовательности байтов, например файл или другое устройство, предоставляющее данные. Потоки позволяют производить операцию чтения и записи информации.

С точки зрения практической реализации ниже приведен пример кода C#, в котором создается строковая переменная path, содержащая путь к файлу с исходными данными. Далее создается поток sr, по которому будет возможно принимать данные из файла, чье имя указано в переменной path. Далее значения переменных s и n считываются непосредственно из файла. По окончанию работы поток закрывается.

Аналогичная реализация в Pascal ABC: вводится строковая переменная path, указывающая на файл. Вводится переменная f, которая ассоциируется с файлом, имя которого указано в path.

C#
```
static void Main(string[] args)
{
string path = "D:\\Мои документы\\Олимпиады\\МИСиС зажигает звезды 2016\\input.txt";
StreamReader sr = new StreamReader(path);
string s = sr.ReadLine();
int n = int.Parse(sr.ReadLine());
sr.Close();
}
```

Pascal ABC
```
var n:integer;
s:string;
f:text;
path:string;
begin
path:='D:\Мои документы\Олимпиады\МИСиС зажигает звезды 2016\input.txt';
Assign(f, path);
reset(f);
readln(f,s);
readln(f,n);
close(f);
end.
```

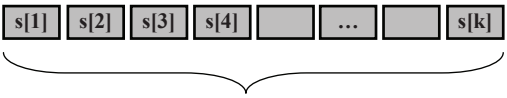
1 этап: формирование вариантов расстановок символов «+» и «-» . При формировании расстановок надо помнить, что необязательно между двумя цифрами должен стоять символ «+» или «-». Наилучший способ разрешить задачу расстановки символов «+», «-» или отсутствия символа – создать строковый массив ins из трех элементов «+», «-», «» (пустая строка) и вставить элементы массива ins в строку s всеми возможными способами.

Pascal ABC
```
var ins:array[0..2] of string;
begin
ins[0]:='';
ins[1]:='+';
ins[2]:='-';
```

C#
```
string[] ins = new string[3];
ins[0] = "";
ins[1] = "+";
ins[2] = "-";
```

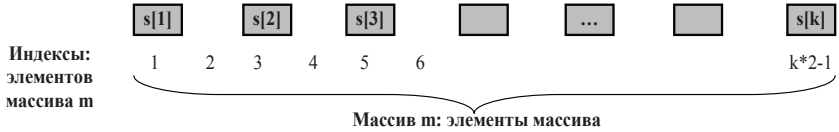
VBA
```
Dim ins(3) As String
ins(1) = ""
ins(2) = "+"
ins(3) = "-"
```

Для вставки элементов массива ins в строку s создадим массив строковый массив m, в который перенесем символы строки s, оставляя между ними пустые позиции для вставки элементов массива ins.



Введенная последовательность чисел s

Тип массива m – string. Размерность массива m равна 2·k-1. Согласно условию задачи k – это количество символов в строке s. Вычислить значение k можно, используя свойство Length, возвращающее длину массива s.



Программная реализация создания массива m и переноса символов строки s в массив m:

Pascal ABC
```
var
m:array of string;
begin
…
k:=s.Length;
m := new
string[2*k-1];
for i:=0 to k-1 do
m[2*i]:=s.Substrin
g(i,1);
…
```

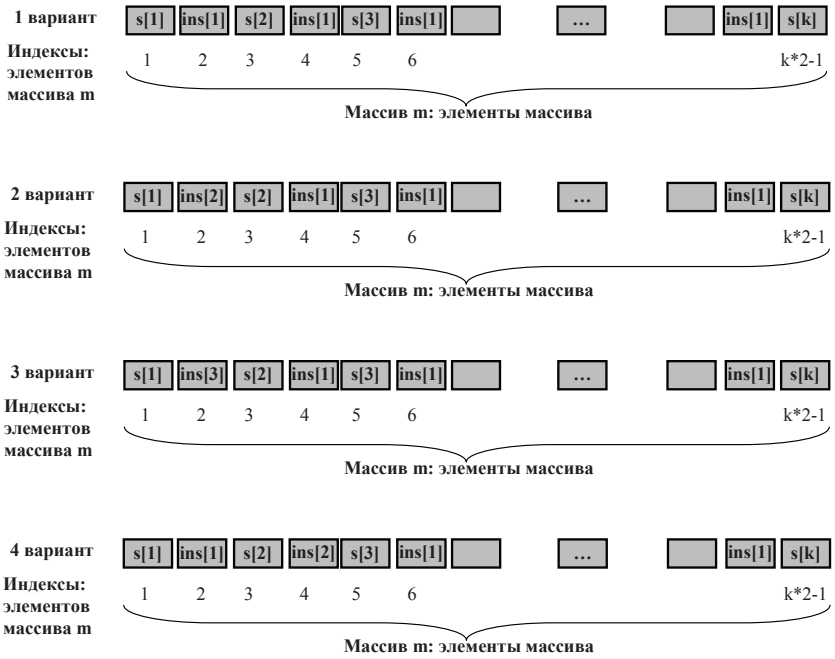
C#
```
string[] m = new
string[2 * k - 1];
for (i = 0; i < k;
i++)
m[2*i] =
s.Substring(i, 1);
```

VBA
```
k = Len(s)
Dim m() As String
ReDim m(k * 2)

For i = 1 To k
m(2 * i - 1) =
Mid(s,i,1)
Next i
```

Стоит обратить внимание на создание и определение длины массива m. Определить длину массива m возможно только после ввода строки s, и, соответственно, после определения ее длины. В C# эта последовательность действий возможна. Но Pascal ABC и VBA не позволяют создавать массив неизвестной длины. Разрешить эту ситуация возможно либо воспользовавшись условием задачи ($1 \le k \le 12$) и создать массив максимально возможной длины, то есть длиной 23. Альтернативный вариант предложен в вышеприведенном коде: создание динамического массива, размер которого может меняться в ходе выполнения кода.

И в Pascal ABC и в VBA массив m создается без указания размера. И только после вычисления значения k определяется размер массива. Также обращаем внимание на несовпадение нумерации массивов в различных реализациях. Это несовпадение вызвано особенностями языка.

Далее незаполненные позиции массива m следует заполнить символами массива ins.



Количество возможных вариантов равно 3 (три элемента массива) в степени (k-1): согласно условию задачи символы «+» и «-» нельзя вписывать до первой или после последней цифры.

Введем переменную q, которая будет определять номер текущей расстановки и будем реализовывать все расстановки в цикле по q:

Pascal ABC	`maxq:=1;`
	`for q:=1 to k-1 do maxq:=maxq*3;`
	`for q:=0 to maxq-1 do`

C#	`for (q = 0; q < Math.Pow(3.0, (double)(k-1)); q++)`
VBA	`For q = 1 To 3 ^ (k - 1)`

В VBA реализация проста: «^» - символ операции возведения в степень.

В C# для возведения в степень использован метод Math.Pow. Параметры метода должны быть вещественными числами, а 3 и (k-1) – числа целого типа; поэтому производится преобразование целого числа к вещественному типу. Более подробно о преобразовании типов написано в рамках решения второго этапа.

В Pascal ABC операция возведения в степень не предусмотрена. Поэтому предварительно рассчитывается 3 ^ (k - 1) через цикл for q:=1 to k-1 do maxq:=maxq*3. Как альтернативный способ расчёта степени в Паскале можно использовать X^Y = exp(ln(X) * Y)

Рассмотрим, как сформировать варианты расстановки:

«» «» «» «» «»
«+» «» «» «» «»
«-» «» «» «» «»
«» «+» «» «» «»
«+» «+» «» «» «»
«-» «+» «» «» «»
….

Заметим, что в каждом варианте расстановки m[2] (первый вставляемый элемент) равен целочисленному расстановки остатку от деления q (номера) на 3

$$m[2] = qMod3 = \left[\frac{q}{3^0}\right] Mod3 .$$

Второй вставляемый элемент – m[4] определяются как целочисленный остаток от деления целой части частного [q/3] на 3

$$m[4] = \left[\frac{q}{3}\right] Mod3 = \left[\frac{q}{3^1}\right] Mod3 .$$

Третий элемент – m[6] определяются как целочисленный остаток от деления целой части частного [q/9] на 3

$$m[6] = \left[\frac{q}{9}\right] Mod3 = \left[\frac{q}{3^2}\right] Mod3 .$$

Анализируя эту цепочку, можно вывести, что $m[j] = \left[\frac{q}{3^{\frac{j-2}{2}}}\right] Mod3 .$

Программная реализация вставки элементов массива ins в массив m производится следующим образом:

Pascal ABC	`maxq:=1;`
	`for q:=1 to k-1 do maxq:=maxq*3;`
	`for q:=0 to maxq-1 do`
	`begin`
	`  for j:=1 to k-1 do`
	`  begin`
	`  st:=1;`
	`  for i:=1 to j-1 do st:=st*3;`
	`  m[j*2-1]:=ins[(q div st) mod 3];`
	`  end;`
	`end;`

C#	`for (q = 0; q < Math.Pow(3.0, (double)(k-1)); q++)`
	`for (j = 1; j < 2*k-1; j=j+2)`
	`m[j] = ins[(int)(q / Math.Pow(3.0, (double)(j-1)/2)) % 3];`

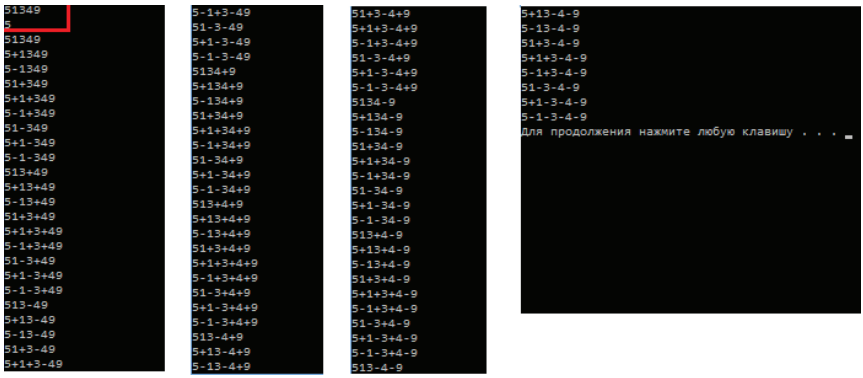
VBA	`For q = 1 To 3 ^ (k - 1)`
	`  For j = 2 To k * 2 - 1 Step 2`
	`  m(j) = ins(1 + (q / 3 ^ ((j - 2) / 2)) Mod 3)`
	`  Next j`
	`Next q`

Для вставки элементов массива ins на 2, 4, 6,…, 2*k-2 позиции массива m в VBA для каждого q расстановка формируется в цикле по j с шагом 2. В C# нумерация массива начинается с 0, поэтому начальное значение j=1. В Pascal ABC шаг равен единице, поэтому, для заполнения каждого второго элемента заполняются элементы m[j*2-1].

Mod – получение целочисленного остатка в VBA и Pascal ABC; в C# для той же цели служит “%”.

Получение целой части от деления в Pascal ABC реализовано оператором div, в C# и в VBA целочисленным делением.

На картинке ниже приведены фрагменты скриншотов вывода всех полученных последовательностей для исходной строки «51349».



2 этап: расчет для расстановки символов результата получившейся последовательности

Основная сложность данного этапа заключается в том, что до сих пор в программной реализации мы имели дело со строковыми данными, а для расчета получившегося выражения необходимо:

1. Выделить слагаемые

2. Произвести математические расчеты (сложение или вычисления в зависимости от знака)

Для решения этой задачи введем:

• переменную числового типа sum, в которой будет накапливаться сумма: sum=sum+<слагаемое> для каждой комбинации;

• переменную sl типа string, в которой будет накапливаться слагаемое: sl=sl+<символьное представление цифры> до тех пор, пока следующим символом не будет «+» или «-»;

• переменную z типа Boolean, которая будет определять знак действия: сложить (true) или вычесть (false).

Далее по очереди будем анализировать элементы массива m: если элемент не «+» и не «-», тогда производим текстовое сложение переменной sl.

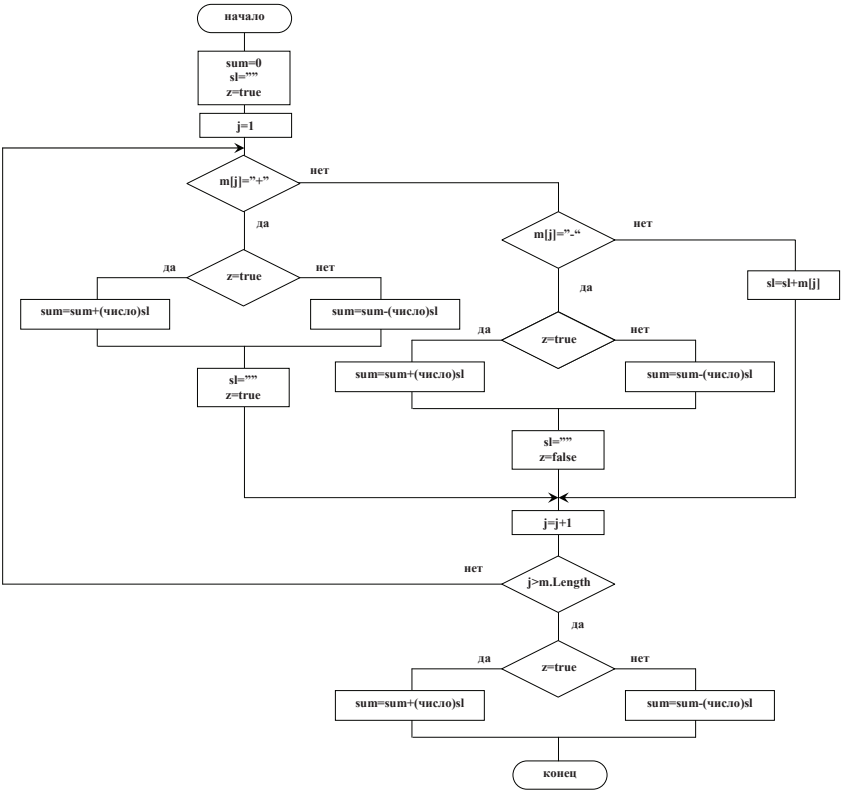
Если элемент «+» или «-», то изменяем переменную sum, добавляя к ней или вычитая из нее накопленное слагаемое в зависимости от значения переменной z, которая показывает знак предыдущего арифметического действия. После чего слагаемое «обнуляется» до пустой строки, а переменная z устанавливается в значение текущего знака.

Данная последовательность действий выполняется до тех пор, пока не будет проанализирован последний элемент массива.

Но, при выходе из цикла важно помнить, что последнее слагаемое не было добавлено (или вычтено). Поэтому по выходу из цикла требуется произвести последнее действие сложение/вычитание.

В приведенной блок-схеме остался необъясненным один момент: переменная sl – строкового типа, а по логике задачи требуется числовое значение. Для решения данного вопроса воспользуемся преобразованием типов данных.

Преобразование типов данных позволяет из данных одного типа (в нашем случае типа string) получить данные другого типа (в нашем случае integer).



Рассмотрим преобразование типов на примере C#:

• Некоторые типы данных могут быть преобразованы «по умолчанию» – без каких-либо действий со стороны программиста. Классическим примером является преобразование типа integer (целое число без дробной части) в тип double (вещественное число с дробной частью). Подобное преобразование может быть выполнено автоматически:

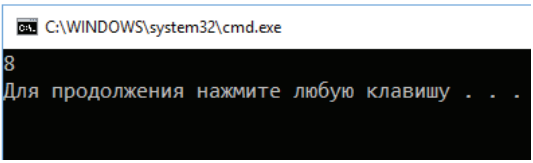
```
static void Main(string[] args)
{
    int i = 5;
    double d = 8.8;
    d = i;
    Console.WriteLine(d);
}
```

```
C:\WINDOWS\system32\cmd.exe
5
Для продолжения нажмите любую клавишу . . .
```

Обратное преобразование (i=d) в данной ситуации невозможно и приведет к ошибке.

• Для преобразования типа double в тип integer требуется явное преобразование, для которого указывается тип, к которому надо привести:

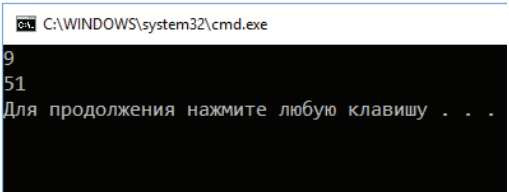
```
static void Main(string[] args)
{
    int i = 5;
    double d = 8.8;
    i=(int)d;
    Console.WriteLine(i);
}
```



При этом возможна частичная потеря данных, как в данном случае была отброшена дробная часть числа.

• Существуют дополнительные встроенные средства преобразования типов данных, в частности `Convert.ToInt32(<что преобразовывать>)` преобразует заданное значение в целочисленный тип

```
static void Main(string[] args)
{
    int i = 5;
    double d = 8.8;
    i = Convert.ToInt32(d);
    Console.WriteLine(i);
    string s = "51";
    i = Convert.ToInt32(s);
    Console.WriteLine(i);
}
```



Из примера видно, что вещественное число d=8.8 преобразуется в целый тип, будучи округлено до числа 9. И строка s=”51” преобразуется в число 51.

В различных языках программирования преобразование типов решается различными способами:

• Как уже было рассмотрено, в C# вычисление суммы будет производиться: sum = sum + Convert.ToInt32(sl);

Convert.ToInt32(sl);преобразует строку sl в числовое значение типа Integer.

• Аналогично в VBA расчет суммы будет производиться как Sum = Sum + CLng(sl)

CLng(sl) преобразует строку sl в число типа Long. Выбран CLng(sl), а не CInt(sl) по причине того, что под переменные типа Integer в VBA отведено всего 2 байта и существует возможность, что

числового диапазона может не хватить для представления числа. Под тип Long отводится 4 байта.

• В Pascal ABC преобразование типа выполняется процедурой val(sl,v,ve), которая принимает входную переменную sl, и возвращает соответствующее ей число v и код результата работы процедуры ve (если преобразование прошло успешно, ve=0, иначе ve>0).

Программная реализация 2 этапа отображена в финальном коде программ.

3 этап: определение в качестве ответа последовательности с наименьшим количеством символов «+» и «-»; если их несколько, то любой из них. Вывод наилучшего результата; при отсутствии вариантов вывод “impossible”.

По окончанию расчета суммы требуется сравнить полученную сумму с числом n. При совпадении – данная последовательность символов будет являться одним из вариантов ответа. Если полный перебор всех последовательностей не обнаружил ни одного ответа, - выводится “impossible”.

Подобная подзадача решается двумя способами:

• Введением числовой переменной sk, которая исходно, до начала поиска последовательностей будет равна 0, а при каждом найденной варианте будет увеличиваться на единицу. Если, по окончанию перебора sk останется равно 0, то ни одного варианта не существует. Данный способ удобен, если необходимо посчитать количество возможных вариантов.

• Введением логической переменной test, которая исходно, до начала поиска последовательности будет равна false, а при нахождении варианта станет равной true. По окончанию перебора данная переменная сигнализирует о наличии/отсутствии найденных вариантов.

Оба способа допустимы. При решении текущей задачи работа логических переменных демонстрируется применением переменной z (знак операции); поэтому для разнообразия и демонстрации применения счетчика в приведенных кодах реализован первый.

Осталось определить, как выбрать и отобразить лучший результат. Существуют (как минимум) два пути решения задачи

• Создать двумерный массив, в который записывать все подходящие последовательности; по окончанию перебора определить, в котором варианте минимальное количество «+» и «-»; вывести этот вариант на экран. Данный способ плох тем, что требуется вводить новый массив (нерациональное использование памяти) и дополнитель-

но перебирать строки данного массива (увеличение времени выполнения программы).

• Ввести строковую переменную res для хранения лучшего варианта расстановки. Исходное значение res=”” (пустая строка).

Для второго варианта требуется ввести еще две переменные kolz (количество символов «+» и «-» в текущем варианте) и kolz_min (минимальное количество символов «+» и «-» в подходящем варианте расстановки символов на текущий момент).

Начальное значение kolz_min – длина строки s (большего количества «+» и «-» быть не может) определяется до перебора всех расстановок.

Значение kolz рассчитывается для каждого варианта расстановки; начальное значение kolz=0 устанавливается до начала расчета суммы варианта расстановки символов.

В процессе вычисления суммы каждой последовательности подсчитывать kolz. Если сумма совпадает с числом n, то сравнивать kolz с kolz_min. Если kolz< kolz_min, следовательно, найден лучший вариант. При этом переменной kolz_min присвоить значение kolz, переменную res обнулить (убрать предыдущие данные) лучший вариант расстановки перенести в переменную res следующим образом: res=res+m[i].

Программная реализация 3 этапа отображена в финальном коде программ.

C#

```
static void Main(string[] args)
{
   int i, j, q, sum, sk, kolz, kolz_min;   //описание переменных
   string sl;                               //
   string res;                              //
   string s = Console.ReadLine();          //ввод числовой последовательности
   int n = int.Parse(Console.ReadLine());  //ввод числа n
   int k = s.Length;                       //определение длины последовательности
   string[] m = new string[2 * k - 1];      //создание массива m
   for (i = 0; i < k; i++)                  //заполнение массива a
        m[2 * i] = s.Substring(i, 1);      //  элементами последовательности s
   string[] ins = new string[3];           //создание массива вставляемых символов
   ins[0] = "";                             //заполнение массива ins
   ins[1] = "+";                            //
   ins[2] = "-";                            //

   sk = 0;                    //обнуление счетчика подходящих вариантов расстановки
   kolz_min = m.Length;          //определение начального значения kolz_min
   res = "";                     //

for (q = 0;q<Math.Pow(3.0, (double)(k - 1)); q++)//формирование вариантов расстановок
    {
```

```
    for (j = 1; j < 2 * k - 1; j = j + 2)//и запись их в массив m
       m[j] = ins[(int)(q / Math.Pow(3.0, (double)(j - 1) / 2)) % 3];

    sum = 0; sl = "";          //2 этап: определение результата каждой расстановки
    bool z = true;                          //обнуление переменных
    kolz = 0;                               //

    for (j = 0; j < m.Length; j++)      //перебор массива m
    {
       if (m[j] == "+")      //если встретился символ "+", то произвести операцию
         {
           if (z)                       //в зависимости от предыдущего знака
                sum = sum + Convert.ToInt32(sl);    //сложение или
            else
                sum = sum - Convert.ToInt32(sl);    //вычитание
            sl = "";                    //слагаемое обнулить
           z = true;                  //знак будущей операции определить как "+"
            kolz++;                     //количество символов увеличить на 1
      }
          else
        {
           if (m[j] == "-")  //если встретился символ "-", то произвести операцию
            {
              if (z)                   //в зависимости от предыдущего знака
                  sum = sum + Convert.ToInt32(sl);   //сложение или
               else
                  sum = sum - Convert.ToInt32(sl);    //вычитание
               sl = "";                 //слагаемое обнулить
              z = false;              //знак будущей операции определить как "-"
               kolz++;                  //количество символов увеличить на 1
            }
           else
  sl = sl + m[j];//если встретился любой иной символ, то дописать его к слагаемому
        }
    }
     if (z)                             //по окончанию перебора массива
         sum = sum + Convert.ToInt32(sl);//добавить
     else
         sum = sum - Convert.ToInt32(sl);//или вычесть последнее слагаемое
                                         //завершение 2 этапа
    if (sum == n)               //если вычисленная сумма равна введенному числу
     {
sk = sk + 1;                      //увеличить счетчик найденных вариантов и
if (kolz < kolz_min)             // в случае, если количество символов "+" и "-"
        {                          // меньше текущего минимального
           res = "";                    //обнулить результат
          kolz_min = kolz;            //присвоить новое минимальное значение
           for (i = 0; i < 2 * k - 1; i++)
               res = res + m[i];       //записать новый лучший результат
        }
     }
}
if (sk == 0)                            //вывод результата
    Console.WriteLine("impossible");
else
    Console.Write(res);
```

```
    Console.WriteLine();
}
```

VBA

```
Sub task()
Dim s As String                 'описание переменных
Dim k As Integer                '
Dim q As Integer                '
Dim z As Boolean                '
Dim n As Integer                '
Dim res As String               '
Dim kolz As Integer             '
Dim kolz_min As Integer         '

s = InputBox("Введите число")   'ввод числа и последовательности
n = InputBox("Введите, чему должна быть равна последовательность")
k = Len(s)                      'определение длины последовательности
Dim m() As String               'создание динамического массива m
ReDim m(k * 2)                  'определение размера массива m

For i = 1 To k                  'заполнение массива m
m(2 * i - 1) = Mid(s, i, 1)     ' элементами последовательности s
Next i

Dim ins(3) As String            'создание массива вставляемых символов ins
ins(1) = ""                     'заполнение массива ins
ins(2) = "+"                    '
ins(3) = "-"                    '

kolz_min = k                    'определение начального значения kolz_min
sk = 0                  'обнуление счетчика подходящих вариантов расстановки

For q = 1 To 3 ^ (k - 1)         'формирование вариантов расстановок
    For j = 2 To k * 2 - 1 Step 2
    m(j) = ins(1 + (q / 3 ^ ((j - 2) / 2)) Mod 3)  'и запись их в массив m
    Next j

    Sum = 0             '2 этап: определение результата каждой расстановки
    sl = ""               'обнуление переменных
    z = True              '
    kolz = 0              '

    For j = 1 To 2 * k   'перебор массива m
        If m(j) = "+" Then'если встретился символ "+", то произвести операцию
            If z Then     'в зависимости от предыдущего знака
            Sum = Sum + CLng(sl)    'сложение
            Else
            Sum = Sum - CLng(sl)    'или вычитание
            End If
            sl = ""                 'слагаемое обнулить
           z = True               'знак будущей операции определить как "+"
            kolz = kolz + 1         'количество символов увеличить на 1
        ElseIf m(j) = "-" Then      'если символ "-", то произвести операцию
            If z Then               'в зависимости от предыдущего знака
            Sum = Sum + CLng(sl)    'сложение
            Else
            Sum = Sum - CLng(sl)    'или вычитание
```

```
            End If
            sl = ""                    'слагаемое обнулить
           z = False                 'знак будущей операции определить как "-"
            kolz = kolz + 1           'количество символов увеличить на 1
        Else
         sl = sl + m(j)'если любой иной символ, то дописать его к слагаемому
        End If
    Next j
            If z Then                  'по окончанию перебора массива
            Sum = Sum + CLng(sl)      'добавить
            Else
            Sum = Sum - CLng(sl)      'или вычесть последнее слагаемое
            End If                     'завершение 2 этапа
    If Sum = n Then            'если вычисленная сумма равна введенному числу
        sk = sk + 1              'увеличить счетчик найденных вариантов и
       If kolz < kolz_min Then 'в случае, если количество символов "+" и "-"
           res = ""        'меньше текущего минимального, обнулить результат
            kolz_min = kolz      'присвоить новое минимальное значение
               For i = 1 To 2 * k
               res = res + m(i) 'записать новый лучший результат
               Next i
            End If
    End If

Next q
If sk = 0 Then MsgBox ("impossible") Else MsgBox (res)      'вывод результата

End Sub
```

Pascal ABC

```
var i,j,q,sum,sk,n,k,maxq,st, kolz,kolz_min:integer; //описание переменных,
sl,s,res:string;                                      //
z:boolean;                                            //
m:array of string;                                   //динамического массива m,
ins:array[0..2] of string;                           //массива символов вставки
v,ve:integer;                                         //
begin
readln(s);                                 //ввод числовой последовательности
readln(n);                                //и числа n
k:=s.Length;                             //определение длины последовательности
m := new string[2*k-1];                 //определение размера массива m

for i:=0 to k-1 do                         // заполнение массива m
m[2*i]:=s.Substring(i,1);                  //элементами последовательности s

ins[0]:='';                                //заполнение массива ins
ins[1]:='+';                               //
ins[2]:='-';                               //

maxq:=1;                                    //вычисление количества вариантов
for q:=1 to k-1 do maxq:=maxq*3;           // расстановок

sk:=0;                  //обнуление счетчика подходящих вариантов расстановки
kolz_min:=k;              //определение начального значения kolz_min
for q:=0 to maxq-1 do                    //формирование вариантов расстановок
begin
  for j:=1 to k-1 do                        //
```

```
  begin                                  //
    st:=1;                               //
    for i:=1 to j-1 do st:=st*3;         //расчет степени тройки
  m[j*2-1]:=ins[(q div st) mod 3];       //запись вариантов в массив m
  end;                                   //
                        //2 этап: определение результата каждой расстановки
  sum:=0;                              //обнуление переменных
  sl:='';                              //
  z:=true;                             //
  kolz:=0;                             //
    for j:=0 to m.Length-1 do          //перебор массива m
    begin                              //
      if m[j]='+' then                 //если встретился символ "+", то
          begin                        //
          val(sl,v,ve);                //преобразовать строку sl в число v и
            if z then                  //произвести операцию в зависимости
              sum:=sum+v               // от предыдущего знака сложение или
              else                     //
              sum:=sum-v;              //вычитание
          sl:='';                      //слагаемое обнулить
          z:=true;                     //знак будущей операции определить как "+"
          kolz:=kolz+1;                //количество символов увеличить на 1
          end
      else
        if m[j]='-' then               //если встретился символ "-", то
          begin                        //
          val(sl,v,ve);                //преобразовать строку sl в число v и
              if z then                //произвести операцию в зависимости
              sum:=sum+v               //от предыдущего знака сложение или
              else                     //
              sum:=sum-v;              //вычитание
          sl:='';                      //слагаемое обнулить
          z:=false;                   //знак будущей операции определить как "-"
          kolz:=kolz+1;                //количество символов увеличить на 1
          end                          //
        else                           //
          sl:=sl+m[j];                 //если встретился любой иной символ,
    end;                               // то дописать его к слагаемому
              val(sl,v,ve);            //по окончанию перебора массива
              if z then                //
              sum:=sum+v               //добавить
              else
              sum:=sum-v;              //или вычесть последнее слагаемое
                                       //завершение 2 этапа
  if sum=n then                      //если вычисленная сумма равна введенному числу
      begin
      sk:=sk+1;                        // увеличить счетчик найденных вариантов и
      if kolz<kolz_min then    // в случае, если количество символов "+" и "-"
        begin                          // меньше текущего минимального
        res:='';                       // обнулить результат
        kolz_min:=kolz;                //присвоить новое минимальное значение
       for i:=0 to 2*k-2 do res:=res+m[i];//записать новый лучший результат
        end;                           //
      end;                             //
end;                                   //
if sk=0 then writeln('impossible') else writeln(res);  //вывод результата
end.
```

ЗАКЛЮЧЕНИЕ

В данном пособии ко всем заданиям приведены решения, в частности 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 авторы относят любой Basic, любой Pascal, любой C).

Учебное издание

Крынецкая Галина Сергеевна
Сидоров Сергей Васильевич

ИНФОРМАТИКА

Методическое пособие по подготовке к олимпиадам школьников

9–11-й классы

В авторской редакции

Компьютерная верстка *И.Г. Иваньшина*

Подписано в печать 01.09.16	Бумага офсетная	
Формат 60 × 90 $^{1}/_{16}$	Печать цифровая	Уч.-изд. л. 4,7
	Тираж 100 экз.	Заказ 5190

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