МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

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

С.В. Сидоров
Г.С. Крынецкая

Методическое пособие
по подготовке к олимпиадам
школьников инженерной
направленности

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

11 класс

МИСиС

Москва 2017

УДК 681.3
К85

Крынецкая Г.С.

К85 Методическое пособие по подготовке к олимпиадам школьников инженерной направленности направленности : Информационно-технологическое направление «Математика в информатике. Решение олимпиадных задач» : 11 класс / Г.С. Крынецкая, С.В. Сидоров. – М. : Изд. Дом НИТУ «МИСиС», 2017. – 71 с.

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

Темы задач – кодирование и представление различных видов информации, различные системы счисления, различные типы данных, алгоритмизация (следование алгоритму, составление алгоритма, анализ алгоритма), программирование и математика. Особое внимание уделено приемам грамотного составления программ, использованию процедур и функций в различных языках программирования. Коды программ приводятся на языках C#, PascalABC и VBA.

Предназначено для подготовки к решению олимпиадных задач, а также для углубленного изучения информатики и математики школьниками выпускных классов школы. Может быть полезно для студентов первых курсов.

УДК 681.3

ОГЛАВЛЕНИЕ

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 7, 8, 9, 10 и 11 классов в два этапа: отборочный и заключительный. Варианты для всех классов различаются по уровню сложности предлагаемых задач.

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

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

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

Это позволяет наиболее заинтересованным читателям потренироваться при решении задач на всех этих языках программирования, сравнить их возможности, почувствовать их преимущества и недос-

татки, убедиться, что все языки структурного программирования в целом похожи.

Пособие предназначено для школьников 11 классов. В нем затрагиваются темы представления информации в памяти компьютера, систем счисления и алгоритмизации. Основное внимание в пособии уделено детальному описанию процесса программирования поставленной задачи и выбору оптимального варианта решения задачи. Пособие может быть полезно студентам первых курсов и учителям информатики.

Задача 1 (Олимпиада школьников МИСиС, 2011 год)

Как представляется число -133_{10} в памяти компьютера?

Проверяется

Знание и понимание представления информации в памяти ЭВМ.

Обсуждается

Прямой, обратный и дополнительный коды, их применения в логике работы процессора.

Теоретическое сопровождение темы. Решение

Известный факт, что в одном байте (8 бит) размещается 255 чисел (комбинации от 0000 0000 до 1111 1111). Если под представление (хранение) числа, согласно условию задачи выделяется один байт, то будут задействованы все 8 бит.

Например, для представления чисел 16 и 32 в памяти ЭВМ, несмотря на то, что число 16 требует 5 разрядов, а число 32 – 6 разрядов, отводится одинаковое количество разрядов.

Естественно, имеется ограничение в количестве представляемых чисел.

Особенности организации памяти ЭВМ таковы, что для представления каждого числа отводится заранее определенное количество байтов в зависимости от его типа.

Например, в Паскаль под число, заявленное как тип Byte, отводится только 1 байт, и, как результат, переменная типа Byte не может хранить значение большее 255. Переменная типа Word размером 2 байта может хранить число от 0 до 65535 (2 в 16 степени). Аналогичные типы данных присутствуют в Бейсике.

Название числового типа данных	Длина, байт числового типа данных	Диапазон значений числового типа данных
Byte	1	0...255
Word	2	0…65535

В Си Шарп имеется большее разнообразие аналогичных типов данных:

Название числового типа данных	Длина, байт числового типа данных	Диапазон значений числового типа данных
byte	1	0..&255
ushort	2	0..&65535
uint	4	0..&4294967295
Ulong	8	0..&2^{64}–1

Возвращаясь к исходному условию задачи, можем кодировать число без знака «133» как «1000 0101» (то есть как двоичный код этого числа). Но в данной задаче надо закодировать число со знаком «-133».

Для возможности представления отрицательных типов один бит выделенных байт (старший) отводится под знак числа: «0» – число положительно; «1» – число отрицательно.

Следовательно, на кодирование абсолютного значения числа отводится на один разряд меньше, что уменьшает максимально возможное кодируемое значение, но сохраняет общее количество кодируемых чисел.

В Паскале и Бейсике это:

Название числового типа данных	Длина, байт числового типа данных	Диапазон значений числового типа данных
ShortInt	1	-128…+127
Integer	2	-32768…+32767
LongInt	4	-2 147 483 648…+2 147 483 647

В Си Шарп это:

Название числового типа данных	Длина, байт числового типа данных	Диапазон значений числового типа данных
sbyte	1	-128…+127
short	2	-32768…+32767
int	4	-2147483648…+ 2147483647
long	8	-2^{63}…+ 2^{63}-1

Таким образом, число «-133» не может быть представлено одним байтом информации и должно представляться типами данных, начиная от Integer или short, то есть минимум – двумя байтами. Но запись числа со знаком не подразумевает, что «-133» будет представлено как «1 000 0000 1000 0101» (единица в старшем разряде указывает на знак, далее 15 бит абсолютного значения). Для представления отрицательных чисел, помимо прямого кода разработаны обратный и дополнительный коды.

Обратный код получается инвертированием числовых разрядов прямого года числа. В частности, число «-133» (код абсолютного значения «0000 0000 1000 0101») в обратном коде будет представле-

но как «1111 1111 0111 1010» (разряд, указывающий на знак, получается единица).

Дополнительный код получается инвертированием и добавлением единицы в младший разряд. В дополнительном коде число «-133» будет представлено как «1111 1111 0111 1011»:

1111 1111 0111 1010

\+

0000 0000 0000 0001

Причина введения обратного и дополнительного кода – при выполнении арифметических операций над числами заменить вычитание сложением. Основным арифметическим узлом в АЛУ процессора является сумматор. Разрабатывая логическую схему сумматора, разработчики пришли к выводу, что проще произвести математические преобразования вычитаемых чисел и заменить вычитание сложением числа в обратном или дополнительном код, чем создать схему «вычитателя».

Продемонстрируем замену вычитания сложением на примере 9–5 = = 4, предполагая, что под каждое число выделяется один байт.

Прямой код числа 9	0 0 0 0 1 0 0 1
Прямой код числа 5	0 0 0 0 0 1 0 1

Преобразуем вычитаемое число 5 как суммируемое -5

Прямой код числа 5	0 0 0 0 0 1 0 1

Для операции сложения двух чисел представим -5 в обратном коде

Обратный код числа -5	1 1 1 1 1 0 1 0

Сложим числа 9 и $-5_{ок}$, используя классические правила сложения чисел

Прямой код числа 9	0 0 0 0 1 0 0 1
Обратный код числа -5	1 1 1 1 1 0 1 0
	1 0 0 0 0 0 0 1 1

Получается, что произошел перенос в непредусмотренный 9ый разряд. По правилам работы с обратным кодом, единицу переноса следует суммировать с младшим разрядом результата.

Прямой код числа 9	0 0 0 0 1 0 0 1
Обратный код числа -5	1 1 1 1 1 0 1 0
	0 0 0 0 0 0 1 1
	1
	0 0 0 0 0 1 0 0

Получившийся результат 0000 0100 соответствует числу 4 в десятичной системе счисления.

Как видно из примера, недостатком обратного кода является двойная операция суммирования. Этот недостаток устраняется введением дополнительного кода.

Прямой код числа 9	0	0	0	0	1	0	0	1
Дополнительный код числа -5	1	1	1	1	1	0	1	1
	1	0	0	0	0	0	1	0	0

Как видно, по результатам суммирования 9 и $-5_{дк}$ также получается единица переноса в 9ый разряд. По правилам суммирования в дополнительном коде эта единица отбрасывается.

Прямой код числа 9	0	0	0	0	1	0	0	1
Дополнительный код числа -5	1	1	1	1	1	0	1	1
	0	0	0	0	0	1	0	0

Результат также получается также 0000 0100.

Ответом данной задачи будет: -133_{10} в одном байте памяти компьютера представляется в обратном коде как «1111 1111 0111 $1010_{2ок}$» или в дополнительном коде как «1111 1111 0111 $1011_{2дк}$».

Типичные ошибки, допущенные при реализации задачи

1. Отсутствие представления о способе представления и хранения чисел.

2. Ответ -1000 0101 ошибочен и содержит сразу две ошибки: выделение под число одного байты (надо 2 байта) и неверное представление знака числа.

Задача 2 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 10 класс)

Вычислите и получите результат в троичной системе счисления $21011_3*9_{10}=$

Проверяется

Знание и понимание систем счисления. Математические навыки.

Обсуждается

Системы счисления. Основание систем счисления.

Теоретическое сопровождение темы. Решение

Наиболее очевидным и наименее удачным вариантом решения данной задачи является преобразование первого множителя из троичной системы в десятичную (получается 193_{10}), умножение его на 9

(получается 1737_{10}) и преобразование результата в троичную систему (получается 2101100_3). Детальное описание данного решения не приводится в силу его очевидности.

Минусом данного решения является значительная потеря времени на математические расчеты и большая вероятность ошибки.

Другим вариантом является преобразование второго множителя в троичную систему $9_{10} = 100_3$.

Естественно, далее можно решить данную задачу, применяя классическое умножение «в столбик»:

При умножении 21011_3 на 100_3 (далее указание на троичную систему опускаем) требуется 21011 умножить на число в последнем разряде множителя (то есть на 0)	21011 * 100 00000
Далее умножить 21011 на следующий по старшинству разряд множителя (то на на 0) и записать результат умножения со сдвигом налево	21011 * 100 00000 00000
Далее умножить 21011 на следующий по старшинству разряд множителя (то на на 0) и записать результат умножения со сдвигом налево	21011 * 100 00000 00000
Далее умножить 21011 на следующий по старшинству разряд множителя (то на на 1) и записать результат умножения со сдвигом налево	2101 * 10 0000 00000 21011
После чего требуется сложить три полученных числа, что (учитывая две строки нулей) крайне несложно.	21011 * 100 00000 + 00000 21011 2101100

Данное решение требует меньше времени на преобразования чисел из одной системы счисления в другую систему, и обратно.

Но, самое краткое и лаконичное решение реализуется при анализе результатов умножения: $21011_3 * 100_3 = 2101100_3$.

Хорошее знание позиционных систем счисления подсказывает, что умножение на основание системы счисления производится переносом запятой (отделяющую целую часть от дробной) на одну позицию вправо. Например, в десятичной системе счисления

98*10 = 980

100*10 = 1000.

Последний пример работает и в двоичной (и во всех прочих) системе счисления:

4*2 = 8 ($4_{10} = 100_2$, $2_{10} = 10_2$, $8_{10} = 1000_2$, $100_2 * 10_2 = 1000_2$)

И при делении на основание системы счисления требуется перенести запятую (отделяющую целую часть от дробной) на одну позицию влево. Это правило работает как в десятичной системе (например, 12300/10 = 1230), так и во всех прочих системах счисления.

Таким образом, для умножения 21011_3 на 100_3 достаточно дописать два нуля справа у первого множителя. Результат $2101\mathbf{100}_3$.

Ответ: 2101100_3.

Типичные ошибки, допущенные при реализации задачи

1. Поиск решения переводом чисел в десятичную систему счисления, умножения в десятичной системе счисления и переводом результата в троичную систему счисления. При подобном подходе возрастает вероятность математических ошибок и тратится слишком много времени.

2. Арифметические ошибки при выполнении деления в двоичной системе счисления. $10011000_2 / 100_2$. Данный вариант решения задачи возможен и не является ошибкой при соблюдении общепринятых правил деления «в столбик».

Задача 3 (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.

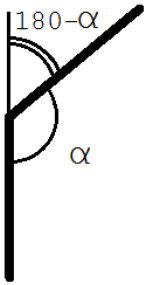
Обсуждается

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).

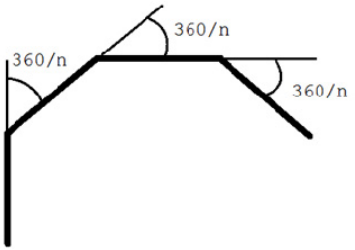


Следовательно, каждый угол правильного n-угольника α равен 180*(n-2)/n.

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

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

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

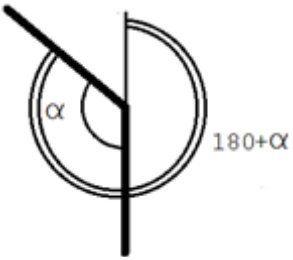
Повтори n-1 [Вперед A Направо 360/n]

Вперед A

Или полностью ему аналогичный по количеству выполняемых команд

Вперед А
Повтори n-1 [Направо 360/n Вперед А]
Оба ответа равносильны и верны.
Вариант ответа
Повтори n [Вперед А 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

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

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

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;
- должно учитываться время выполнения предлагаемой программы (согласно условию предполагается минимальное число шагов);
- должно учитываться понимание влияния параметров на выполнение программы;
- учитывается понимание допустимых значений параметров.

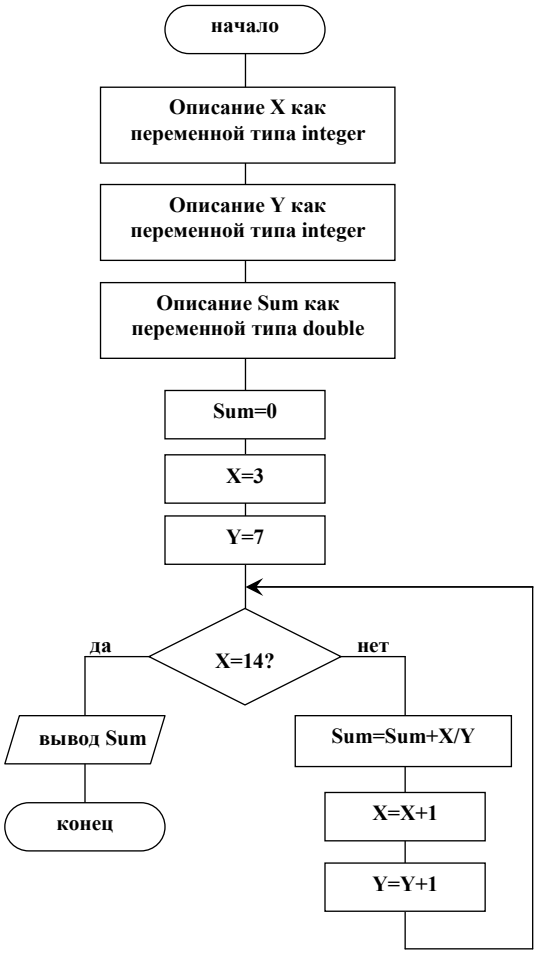
Типичные ошибки, допущенные при реализации задачи

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

При работе программы, написанной по нижеприведенному алгоритму, реализованной в C#, результатом работы программы является число 0.

Дайте объяснения причине неверных расчётов. Сделайте исправления в алгоритме, чтобы расчет был верен. Напишите, что должна делать программа, согласно заданному алгоритму (вычислять точный ответ необязательно).



Проверяется

Навыки алгоритмизации, умения работать с блок-схемами, анализировать алгоритмы, знание типов данных, навыки поиска ошибок.

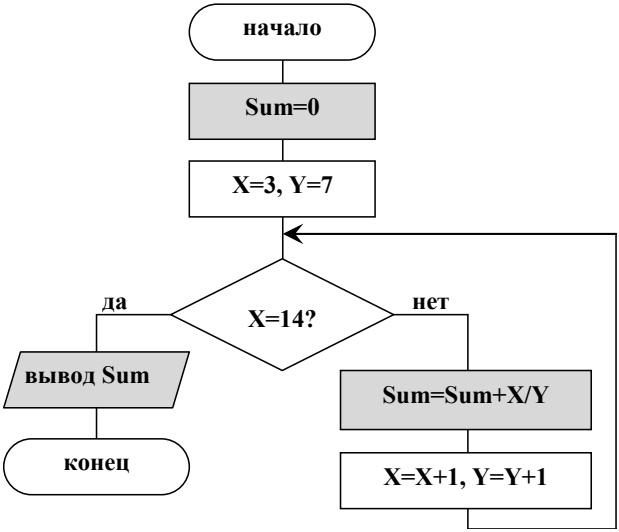
Обсуждается

Типы данных. Анализ алгоритма. Допустимые операции над различными типами данных и их реализация. Преобразование типов данных.

Теоретическое сопровождение темы. Решение

На первый взгляд данная задача проста: блок-схема корректно реализует классический алгоритм суммирования. Исходно сумма обнуляется, в цикле происходит накопление суммы путем добавления к значению суммы нового слагаемого, по выходу из цикла производится вывод результата на печать (на рисунке эти блоки отображены серым цветом).

Упрощенный фрагмент блок-схемы



Анализ переменной X, определяющей вход в цикл и выход из цикла показывает, что

- исходное значение X равно 3, цикл работает до тех пор, пока X не станет равен 14; следовательно, вход в цикл возможен; следовательно, операция накопления суммы должна выполняться.
- при каждом входе в цикл X увеличивает свое значение на единицу, следовательно, в цикл программа должна войти 11 раз (начиная с X = 3 и до последнего входа при X = 13)
- значение переменной X = 14 при подобной реализации цикла достигается при одиннадцатом входе в цикл, следовательно, 12-го входа в цикл быть не должно; цикл останавливается, программа благополучно завершается.

Последнее наблюдение подтверждается условием задачи: «результатом работы программы является число 0», то есть программа выполняет команду вывода результата суммирования на экран.

Дополнительный анализ цикла показывает, что

- значение суммы должно увеличиваться на величину X/Y;
- исходное значение X = 3, Y = 7, следовательно, при первом входе в цикл значение суммы должно быть рассчитано как 0+3/7;
- при каждом входе в цикл после суммирования увеличивается на единицу как значение X, так и значение Y; следовательно, на каждом следующем шаге изменяется как числитель прибавляемой дроби, так и знаменатель;
- последний вход в цикл производится при X = 13, следовательно, программа должна рассчитывать сумму ряда 3/7+4/8+5/9+6/10 + + 7/11 + 8/12+9/13+10/14+11/15+12/16+13/17;
- при черновом анализе результат работы программы должен быть дробным числом больше нуля (все слагаемые положительны и отличны от нуля) и меньше 11-ти (все слагаемые меньше единицы).

Но, согласно условию, на экран выводится нулевое значение результата, то есть складывается ситуация, когда при корректной организации цикла получается неверный ответ.

Рассмотренный фрагмент блок-схемы не содержит ошибок. Вернемся к начальному фрагменту блок-схемы:

Данный фрагмент содержит описание переменных, используемых в программе:

- переменные X и Y типа integer;
- переменная Sum типа double.

Переменные типа integer (в некоторых языках данный тип называется int) предназначены для хранения целых чисел определенного размера. Переменные типа double предназначены для хранения чисел с плавающей запятой, то есть для дробных чисел.

Тип переменной определяет

- количество памяти, выделенное для данной переменной, которое влияет на максимальную величину значения переменной;
- допустимые операции с переменной;
- порядок выполнения операции над переменными указанного типа.

Например, в VBA под переменную целого типа выделается 2 байта памяти. В 2 байта (16 бит) можно записать числовые значения от 00000000 00000000 до 11111111 11111111 (пробелы ставятся для удобства восприятия). Если бы данный тип переменных использовался для чисел без учета знака, до минимально-возможным числом являлось бы 0, а максимальным – 65535 (равное 2^{16}-1). Но тип integer работает как с положительными, так и с отрицательными числами. Для учета знака старший бит числа отводится под знак (если он равен 0 – то число положительно, если он равен 1 – число отрицательно). Следовательно, диапазон представления чисел типа integer в VBA – от -2^{15} до $+2^{15}$, то есть от -32768 до +32767. В C# под переменную типа int отводится 4 байта, что позволяет работать с числами от -2147483648 до +2147483647 (то есть от -2^{32} до $+2^{32}$-1).

Под переменные типа double для представления вещественных чисел в любом языке программирования отводится существенно больше памяти (порядка восьми байт). Причиной тому является как множество символов (после запятой), так и принцип хранения этих чисел. Вещественное число приводится к нормализованному виду: например, 34,67 в нормализованном виде записывается как $0{,}3467*10^2$. 3467называется мантиссой, а 2 (степень при 10) – порядком числа. В памяти отводится разряды как для хранения мантиссы, так и для хранения порядка числа.

С каждым типом переменных определены допустимые операции: если с числами (как с типом integer, так и с типом double) возможны арифметические операции (сложение, умножение, возведение в степень и проч.), то со строками (переменные типа string) не предусмотрены ни операции деления, ни возведения в степень.

Допустимые операции могут по-разному выполняться для различных типов данных. Примером может служить операция сложения, которая определена как для чисел, так и для строк. Но при сложении чисел 12 и 34 в результате получается число 46, а при сложении строк «12»и «34» в результате получается строка «1234».

Рассмотрим, как выполняется вычисление Sum=Sum+X/Y с учетом того, что X и Y – переменные целых типов, а Sum – переменная вещественного типа.

Операция деления для целых чисел определена, но при делении целого числа на целое число без дополнительных команд преобразования типа, результатом должно быть целое число. Например, при делении числа 14 на 5 в результате будет число 2 – целая часть деления (14/5=2,8).

Таким образом, при делении целого числа 3 на целое число 7 результатом вычисления первого слагаемого будет ноль. Аналогично второе слагаемое, равное 4/8 будет вычислено также равное нулю. Все слагаемые заданного ряда чисел, имея числитель меньший, чем знаменатель, будут также равны нулю.

По завершению цикла сумма останется равной нулю.

Причина ошибки – целочисленное деление чисел типа integer, дающее в результате ноль.

C# в данной задаче указан как один из языков программирования, который делит целое число на целое число, получая, в соответствии со своей встроенной логикой, в ответе только целую часть числа, то есть ноль. Многие языки программирования аналогично реализуют данный алгоритм. Эта информация в задаче не является принципиальной.

Принципиальным в нахождении правильного ответа является умение анализировать алгоритмы, и знание типов данных; особо важно понимание различия в выполнения одноименных операций для разных типов данных.

Для устранения данной ошибки есть несколько возможностей. Самым очевидным ответом является изменение типа переменных X и Y. Если X и Y описать как переменные вещественного типа (float или уже упомянутый double), то деление X/Y будет давать в результате вещественное число, которое будет суммироваться к Sum.

Стоит сделать примечания для программирующих в C#: при присвоении значений X и Y требуется присваивать значения 3.0 и 7.0 (а не 3 и 7), записывая присваемые значения в вещественном формате.

Но не всегда логика задачи допускает изменение типа переменных. Кроме того, стоит иметь в виду, что переменные вещественного типа занимают больше места в оперативной памяти. Альтернативный способ исправления ошибки – преобразовывать значения целочисленных переменных X и Y в вещественный формат значения непосредственно в момент деления: (double)X/(double)Y.

Операции преобразования типа предусмотрены практически во всех языках. Но стоит помнить, что не всегда возможно корректное преобразование типа данных. Например, преобразование типа integer в тип double не составляет особой сложности: добавить разряды для преобразования целого числа в дробное можно. При обратном пре-

образовании дробного числа в целое требуется «убирать» «лишние» разряды, оставляя только целую часть, с частичной потерей точности числа, что может привести к ошибке расчетов.

Хотя данная задача не требует в ответе программной реализации, в рамках объяснения темы типов данных, вариантов преобразования типов данных, а также для демонстрации возможностей языков программирования при реализации различных вариантов решения задачи приведем варианты решения данной задачи в различных языках программирования.

Первый код реализован на языке C# и содержит вариант с ошибкой: сумма равна нулю.

```
static void Main(string[] args)
   {
    int x = 3;       //описание x как переменной типа integer и
                              //присваивание ей значения 3
    int y = 7;       //описание y как переменной типа integer и
                              //присваивание ей значения 7
    double s = 0;    //описание s как переменной типа double и
                              //присваивание ей значения 0
    for (x = 3; x < 14; x++)    //цикл по счетчику, меняющий
     {               //значения x от 3 до 13 и шагом 1 (x++)
         s = s + x / y;          //суммирование s
         y++;                    //увеличение y на единицу
     }
     Console.WriteLine("Сумма равна {0} ", s);   //вывод результата s
   }
```

В рамках исправления предложенного кода можно

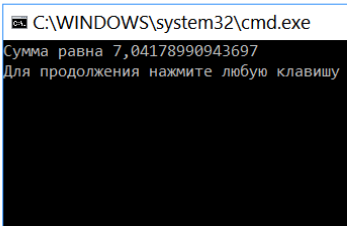
- изменить исходный тип переменных:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double x = 3;
            double y = 7;
            double s = 0;
            for (x= 3;  x< 14; x++)
            {
                s = s + x/y;
                y++;
            }
            Console.WriteLine("Сумма равна {0} ",s);

        }
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
Сумма равна 7,04178990943697
Для продолжения нажмите любую клавишу
```

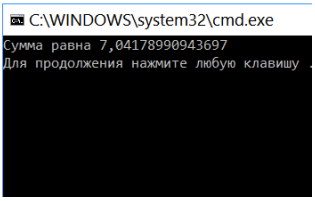
- привести переменные к вещественному типу в процессе деления, что в C# можно реализовать либо так:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 3;
            int y = 7;
            double s = 0;
            for (x= 3;  x< 14; x++)
            {
                s = s + (double)x/(double)y;
                y++;
            }
            Console.WriteLine("Сумма равна {0} ",s);

        }
    }
}
```



либо так:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 3;
            int y = 7;
            double s = 0;
            for (x= 3;  x< 14; x++)
            {
                s = s + Convert.ToSingle(x) / Convert.ToSingle(y);
                y++;
            }
            Console.WriteLine("Сумма равна {0} ",s);

        }
    }
}
```



Как видно из приведенных скриншотов, все варианты исправляют ошибку.

Также при рассмотрении кода видно, что в C# допускает одновременное описание переменных и присвоение им значений в рамках программы до первого использования.

Реализация цикла по счетчику for в C# позволяет предложенный алгоритм немного модифицировать и сделать более лаконичным:

```
static void Main(string[] args)
   {
     int x;            //описание x как переменной типа integer
     int y;           //описание y как переменной типа integer
     double s = 0;       //описание s как переменной типа double и
                         //присваивание ей значения 0
     for (x = 3,y=7; x < 14; x++,y++)   //цикл по счетчику,
       {
       s = s + (double)x / (double)y;        //суммирование s
       }
```

```
        Console.WriteLine("Сумма равна {0} ", s);   //вывод результата s
    }
```

В модифицированном варианте начальные значения переменных X и Y, а также – их модификация (изменение после каждого выполненного шага цикла) задаются непосредственно в цикле for. Более того, описание переменных также можно реализовывать непосредственно в цикле:

```
static void Main(string[] args)
        {
            double s = 0;
            for (int x = 3, y=7; x < 14; x++,y++)
            {
                s = s + (double)x / (double)y;
            }
            Console.WriteLine("Сумма равна {0} ", s
        }
```

Коды в Pascal ABC и в VBA реализуют приведенный алгоритм, но при этом внутренняя логика языков Pascal ABC и VBA при делении целого числа на целое число позволяет получить дробный результат, а потому выполнять преобразование типа в данном случае не требуется.

Pascal ABC:

```
var s:double; //описание s как переменной типа double
x,y:integer;  //описание x и y как переменных типа integer
begin
y:= 7;        //присваивание переменной y значения 7
s:= 0;        //присваивание переменной s значения 0
for x:= 3 to 13 do //цикл по счетчику от x=3 до x=13 с шагом 1
begin
s:= s + x/y;  //накопление суммы
y:= y+1;      //увеличение значение y на 1
end;
writeln(s);    //вывод результата s
end.
```

Особенностью реализации цикл по счетчику for в Pascal ABC является то, что шаг всегда либо увеличивается, либо уменьшает на единицу.

VBA:

```
Sub summa()
Dim x As Integer      'описание x как переменной типа integer
Dim y As Integer      'описание y как переменной типа integer
Dim s As Double       'описание s как переменной типа double
y = 7                  'присваивание значения 7 переменной y
s = 0                  'присваивание значения 0 переменной s
For x = 3 To 13 step 1 'цикл по счетчику от x=3 до x=13 с
шагом 1
```

```
s = s + x / y              'накопление суммы
y = y + 1                  'увеличение значение y на 1
Next x
MsgBox (s)                 'печать результата
End Sub
```

Особенностью реализации цикл по счетчику for в VBA является, что при увеличении счетчика на единицу шаг можно не указывать и строка «For x = 3 To 13 step 1» может быть записана как «For x = 3 To 13».

Ответ: программа подсчитывает ряд сумму ряда чисел: 3/7+4/8 + + 5/9+6/10+7/11+8/12+9/13+10/14+11/15+12/16+13/17. Причина ошибки: целочисленное деление чисел типа integer, дающее в результате ноль. Для исправления ошибки надо

- либо описывать переменные X и Y как переменные вещественного типа;
- либо при делении приводить переменные X и Y к вещественному типу.

Типичные ошибки, допущенные при реализации задачи

1. Незнание типов данных и особенностей операций с различными типами данных.

2. Непонимание алгоритма.

3. Пренебрежение частью ответа. В данной задаче ответ состоит из трех компонент: определить, что именно должен делать алгоритм, определить ошибку алгоритма, исправить ошибку.

4. Неверное определение количества слагаемых: 3/7+4/8+5/9+ +6/10+7/11+8/12+9/13+10/14+11/15+12/16+13/17**+14/18** ответ неверный.

Задача 5 (Олимпиада по информатике «МИСиС зажигает звезды», 2016 год, 10 класс, финальный тур)

Напишите программу (нарисуйте блок-схему, приведите алгоритм), которая реализует работу автомата, входные и соответствующие им выходные данные которого приведены в таблице:

На входе автомата	На выходе автомата
925784, 2, 1	11
925784, 2, 4	9
925784, 5, 6	12
925784, 3, 7	error
925784, 3, 6	9
925784, 6, 6	8
925784, 6, 7	error

Проверяется

Умение выявлять закономерности, представление о функциях и параметрах функций. Типы данных. Умение составлять алгоритмы, строить блок-схемы и программировать.

Обсуждается

Анализ данных. Способы решения задач на выявление закономерностей.

Типы данных и их влияние на реализацию алгоритма. Составление алгоритма и программирование задачи в зависимости от типа входных данных.

Применение в программировании функций (методов, подпрограмм, процедур). Сравнительный анализ кода, написанного с использованием функций и без использования функций.

Наглядная демонстрация программной реализации алгоритма в различных языках программирования (с учётом особенностей языка).

Способы контроля вводимой в программу информации в различных языках программирования.

Теоретическое сопровождение темы. Решение

Перед тем, как приступать непосредственно к написанию программы, при решении данной задачи требуется определить логику работы автомата. Для решения данной и подобной ей задач рекомендуется руководствоваться следующими правилами:

- В исходных данных задачи присутствует информация, необходимая для решения задачи. Иными словами – в задаче приведено достаточное количество входных данных, чтобы подметить закономерность, по которой работает автомат.
- Важно понять, какие входные параметры и почему приводят к ошибкам. В ряде случаев закономерность проще определить, отталкиваясь от ошибочных вариантов входных данных.
- Рассматривать входные и выходные данные, учитывая, что существует множество различных типов данных. Не всегда очевидная информация является единственно верной версией. Например, не всегда ряд символов-цифр надо рассматривать как число. Запись «123456» может являться как числом «сто двадцать три тысячи четыреста пятьдесят шесть», так и строкой из шести символов-цифр «123456».
- Параллельно анализировать результаты при сходных входных параметрах.

Рассмотрим входные параметры и их результаты, чтобы определить недопустимые значения параметров.

На входе автомата	На выходе автомата
925784, 2, 1	11
925784, 2, 4	9
925784, 5, 6	12
925784, 3, 7	Error

925784, 3, 6	9
925784, 6, 6	8
925784, 6, 7	Error

Обе ошибки возникают, когда значение второго параметра равно семи. Причем, значение второго параметра менее семи, является допустимым. Внимательный анализ входных данных показывает, что число семь связано с первым параметром: количество символов в первом параметре равно шести; седьмого символа нет.

Из этого наблюдения можно сделать допущение, что третий параметр указывает на номер символа в первом параметре. В пользу этого подтверждения свидетельствует факт, что значение третьего параметра – целое число, больше нуля. Все безошибочные данных не противоречат данной версии.

Дальнейшее изучение входных параметров показывает, что второй параметр может быть либо больше, либо меньше, либо равен третьему. Можно сделать допущение, что второй параметр также указывает на позицию символа в первом параметре.

Далее разумно рассмотреть сходные значения параметров – в первой и второй строке. А также – в третьей, пятой и шестой.

Разные значения второго и третьего параметров дают различные результаты. Но, например, число 9 на выходе можно получить разными путями – вторая и пятая строка. Примем данный факт к сведению. Вывести из него закономерность не получается.

На входе автомата	На выходе автомата
925784, 2, 1	11
9**25**784, 2, 4	9
9257**84**, 5, 6	12
925784, 3, 7	Error
92**5**78**4**, 3, 6	9
92578**4**, 6, 6	8
925784, 6, 7	Error

Далее жирным шрифтом выделим позиции в первом параметре, номера которых заданы вторым и третьим параметром.

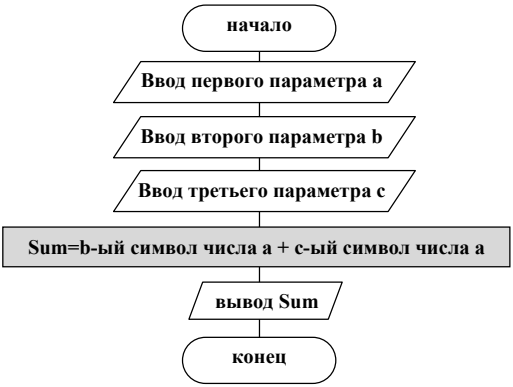
Из рассматриваемых записей надо найти соответствие между числами:

Числа первого параметра в позициях, указанных вторым и третьим	На выходе автомата
9 2	11
2 7	9
8 4	12
5 4	9
4 4	8

Нетрудно увидеть, что число на выходе автомата соответствует сумме цифр из первого параметра, позиции которых указаны вторым и третьем параметром.

Сделанный вывод подтверждается всеми примерами, и – косвенно – ошибками.

Определив, что именно должен делать автомат, реализуем алгоритм работы данного автомата. В первом приближении блок схема будет иметь следующим вид:



При внешней простоте алгоритма его реализация не так проста, как кажется на первый взгляд:

1. Если результат формируется из цифр первого параметра, то все вводимые символы первого параметра обязательно должны быть цифрами.

2. Второй параметр обязательно должен быть целым числом больше нуля, но не больше, чем количество символов первого параметра.

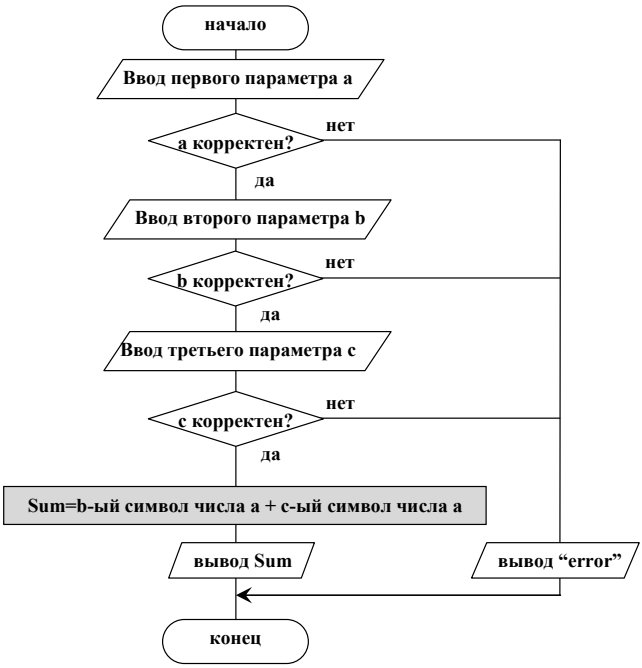
3. Аналогично третий параметр обязательно должен быть целым числом больше нуля, но не больше, чем количество символов первого параметра.

4. Самое существенное затруднение связано с выделенной в блок-схеме операцией суммирования символов числа ***a***:

   a) если ***a*** вводится как число, то требует разработать математический алгоритм извлечения символов с нужной позиции;

   b) если ***a*** вводится как строка (а некоторые языки программирования принимают информацию только в строковом формате), то требуется извлечь символы с нужных позиций и преобразовать их в числа для дальнейшего суммирования.

Следовательно, с учетом контроля вводимых данных, блок-схема может быть представлена следующим образом:



Возвращаясь к вопросу суммирования требуемых символов первого параметра, рассмотрим варианты решения более подробно.

Начнем с варианта, когда первый параметр ***a*** является числом. Рассмотрим следующий набор входных параметров: «745319, 2, 5». Согласно входным данным требуется из числа «семьсот сорок пять тысяч триста девятнадцать» извлечь числа «4» и «1» (со второй и пятой позиции).

Для решения данной математической задачи удобно использовать операции:

- получения целой части от деления (например, результат целочисленного деления 17 на 3 будет 5); в данной задаче, чтобы «отбросить» «лишние» символы справа можно число 745319 целочисленно поделить на 10 несколько раз.
- получения остатка от деления (например, результат остатка от деления 17 на 3 будет 2) можно использовать для «отбрасывания» символов слева от нужного символа.

Например, чтобы выделить число «4» из второй позиции, целочисленно делим 745319 на 10 четыре раза:

- 745319 «целочисленное деление» на 10=74531;
- 74531 «целочисленное деление» на 10=7453;

- 7453 «целочисленное деление» на 10=745;
- 745 «целочисленное деление» на 10=74.

Для получения «4» делим полученное значение на 10 операцией получение целочисленного остатка:

74 «получение остатка от деления» на 10 = 4

Аналогичные действия надо произвести над исходным числом для выделения из него числа «1» (в пятой позиции) с тем отличием, что выделение числа из второй позиции потребовало 4-хкратного целочисленного деления на 10; а выделение числа из пятой позиции требует однократного деления на 10.

Попробуем сформулировать математически, сколько раз надо целочисленно делить исходное число на 10. Мы видим, что, чем меньше номер позиции, тем больше символов справа от нее надо «убрать»:

Позиция числа 745319	Сколько раз делим
6	0 = 6-6
5	1 =6-5
4	2 =6-4
3	3= 6-3
…	

Из приведенного примера можно вывести математическую закономерность: количество делений равно количество цифр в первом параметре минус позиция выделяемого числа.

В различных языках программирования данные операции обязательно присутствуют, хотя реализуются различными способами.

Язык	Получение целой части от деления	Получение остатка от деления

C#	`int a1=a;`
	`a=a/10;`
	получение целой части от деления реализуемся обычным делением при условии, что переменная a описана как целое число
		`a = a - (a / 10) * 10;`
		получение целого остатка от деления реализуемся вычитанием из исходного числа его самого, с предварительно убранным последним разрядом, например (используется целочисленное деление): 745319-(745319/10)*10=
		745319/10 = 74531
		74531*10=745310
		745319 – 745310=9

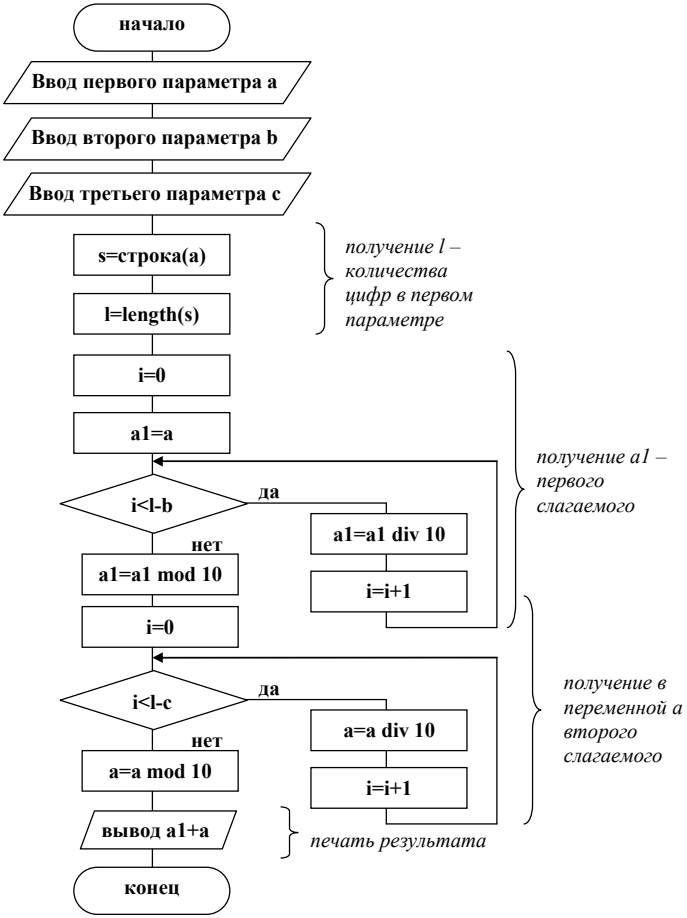
PascalABC	`a:=a div 10;`	`a:= a mod 10;`
VBA	`a = a \ 10`	`a = a Mod 10`

Для определения количества цифр в первом параметре можно использовать различные механизмы:

- целочисленно делить первый параметр на 10 до тех пор, пока остаток не будет равен нулю;

- использовать оператор преобразования числа *a* в строку *s* и оператор определения длины строки *s*; названные операторы обязательно присутствуют в каждом языке.

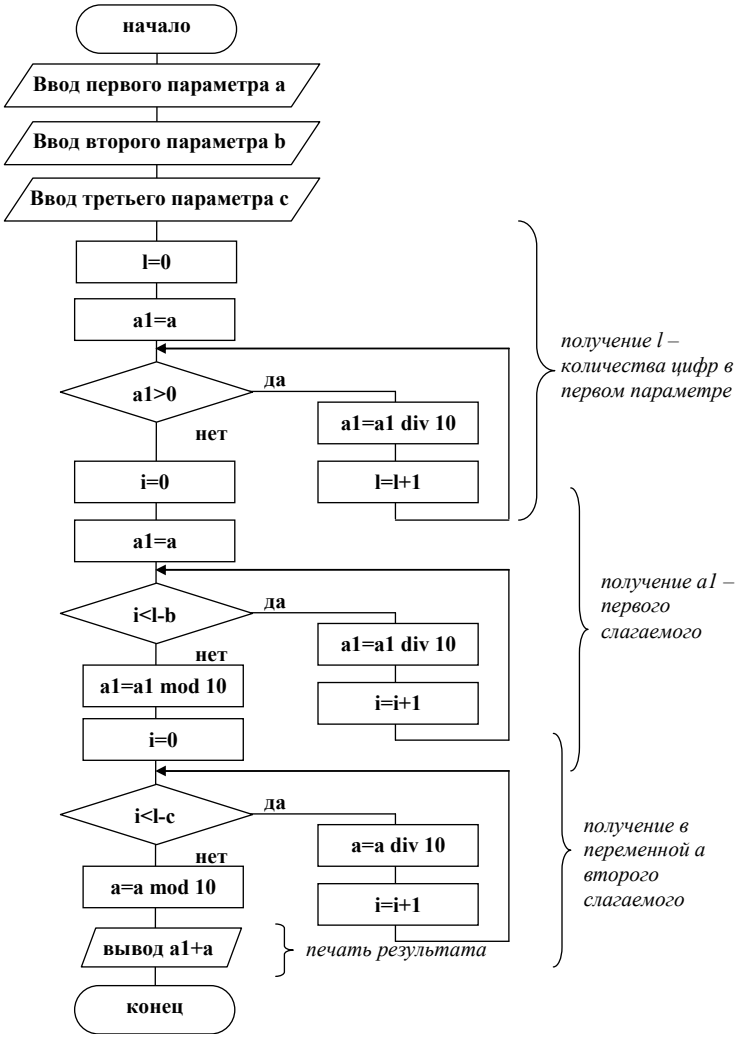
В силу всего вышеизложенного математический алгоритм, в котором длина первого параметра вычисляется через преобразование типов (для простоты рассмотрим его без контроля входных данных) будет выглядеть следующим образом:



До начала выделения первой цифры из первого параметра значение первого параметра (*a*) дублируется в переменную *a1*, чтобы сохранить исходное значение *a* для выделения второй цифры. В процессе мате-

матических преобразований в переменной *a1* должно получиться первое слагаемое. Второе слагаемое формируется в переменной *a*.

Аналогичный алгоритм, в котором длина первого параметра считается математически, приведен ниже. В нем переменная *a* дважды копируется в *a1*: в первый раз для определения ее длины (в завершение значение *a1* станет равным нулю); во второй – для получения в переменной *a1* первого слагаемого.



Рассмотрим программную реализацию без учета ошибок (считаем, что пользователь введет верные параметры) для предложенных вариантов.

C# (с определением длины через преобразование типа):

```
static void Main(string[] args)
        {
         Console.WriteLine("введите первый параметр");
         int a = int.Parse(Console.ReadLine());
         string s = Convert.ToString(a); //преобразование числа a в строку s
         int l = s.Length;               //вычисление длины s
                int b, c;
                Console.WriteLine("введите второй параметр");
                b = int.Parse(Console.ReadLine());
                Console.WriteLine("введите третий параметр");
                c = int.Parse(Console.ReadLine());
            int i;
            int a1 = a;                  //получение первого слагаемого
            for (i = 0; i < l - b; i++)
                a1 = a1 / 10;
            a1 = a1 - (a1 / 10) * 10;
            for (i = 0; i < l - c; i++) //получение второго слагаемого
                a = a / 10;
            a = a - (a / 10) * 10;
      Console.WriteLine("{0}", a1 + a); //печать результата
        }
```

C# (с определением длины через деление):

```
static void Main(string[] args)
{
    Console.WriteLine("введите первый параметр");
    int a = int.Parse(Console.ReadLine());
        int b, c;
        Console.WriteLine("введите второй параметр");
        b = int.Parse(Console.ReadLine());
        Console.WriteLine("введите третий параметр");
        c = int.Parse(Console.ReadLine());
    int i;
    int l;                          //расчет длины a накоплением l
        int a1 = a;
        for (l = 0; a1>0; l++)      //пока число не равно нулю,
            a1 = a1 / 10;           //делим его на 10, увеличивая l
    a1 = a;
    for (i = 0; i < l - b; i++)
        a1 = a1 / 10;
    a1 = a1 - (a1 / 10) * 10;
    for (i = 0; i < l - c; i++)
        a = a / 10;
    a = a - (a / 10) * 10;
    Console.WriteLine("{0}", a1 + a);
}
```

Pascal ABC (с определением длины через преобразование типа):

```
var a,b,c,l,a1,i:integer;
s:string;
begin
      writeln('введите первый параметр');
      readln(a);
      str(a,s);          //преобразование числа a в строку s
      l:=length(s);      //вычисление длины s
writeln('введите второй параметр');
readln(b);
writeln('введите третий параметр');
readln(c);
      a1:=a;
      for i:=1 to l-b do
      a1:=a1 div 10;
      a1:= a1 mod 10;
for i:=1 to l-c do
a:=a div 10;
a:= a mod 10;
writeln(a1+a);
end.
```

Pascal ABC (с определением длины через деление):

```
  var a,b,c,l,a1,i:integer;
  s:string;
  begin
        writeln('введите первый параметр');
        readln(a);
        writeln('введите второй параметр');
        readln(b);
        writeln('введите третий параметр');
        readln(c);
  l:=0;
  a1:=a;
  repeat
  a1:=a1 div 10;
  l:=l+1;
  until (a1=0);
        a1:=a;
        for i:=1 to l-b do
        a1:=a1 div 10;
        a1:= a1 mod 10;
  for i:=1 to l-c do
  a:=a div 10;
  a:= a mod 10;
  writeln(a1+a);
  end.
```

VBA (с определением длины через преобразование типа):

```
Sub avtomat()
Dim a As Long
Dim l As Integer
Dim b As Integer
Dim c As Integer
Dim i As Integer
Dim a1 As Long
```

```
      a = InputBox("Введите первый параметр")
      l = Len(CStr(a)) 'CStr преобразует a в строку; Len – длина строки
b = InputBox("Введите второй параметр ")
c = InputBox("Введите третий параметр")
      a1 = a
      For i = 1 To l - b
      a1 = a1 \ 10
      Next i
      a1 = a1 Mod 10
For i = 1 To l - c
a = a \ 10
Next i
a = a Mod 10
      MsgBox (a1 + a)
End Sub
```

VBA (с определением длины через деление):

```
Sub avtomat()
Dim a As Long
Dim l As Integer
Dim b As Integer
Dim c As Integer
Dim i As Integer
Dim a1 As Long
      a = InputBox("Введите первый параметр")
      l = 0
      a1 = a
      Do
      a1 = a1 \ 10
      l = l + 1
      Loop Until a1 = 0
b = InputBox("Введите второй параметр ")
c = InputBox("Введите третий параметр")
      a1 = a
      For i = 1 To l - b
      a1 = a1 \ 10
      Next i
      a1 = a1 Mod 10
For i = 1 To l - c
a = a \ 10
Next i
a = a Mod 10
      MsgBox (a1 + a)
End Sub
```

Тщательный анализ кода показывает, что для выделения двух цифр (двух слагаемых) надо дважды делить число *a* некоторое количество раз, то есть фактически повторять одно и то же действие. Чтобы избежать дублирования, можно в одном цикле выделить все цифры числа *a* и занести их в массив m для дальнейшего использования.

Если опять рассмотреть входные данные «745319, 2, 5», то массив m надо заполнить цифрами «7», «4», «5», «3», «1» и «9». Причем удобнее (хотя и не обязательно), заполнять массив с первого элемента, сохраняя порядок цифр исходного числа:

m[1]=7;
m[2]=4;
m[3]=5;
m[4]=3;
m[5]=1;
m[6]=9.

В этом случае результат получается как m[b]+m[c].

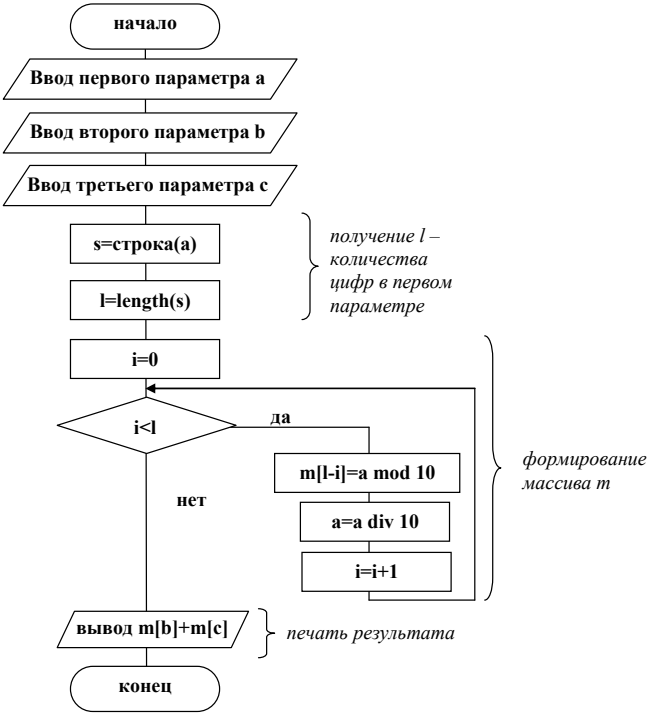
Однако существует некоторое затруднение, связанное с подобным формированием массива: по математической логике уже рассмотренных (и реализованных) циклов цифры первого из параметра извлекаются с конца: первой выделенной цифрой будет «9» (которая должна быть записана в m[6]), второй – «1» (которая должна быть записана в m[5]), третьей – «3» (которая должна быть записана в m[4]), и т.д.

Следовательно, массив должен заполняться с конца: на первом шаге заполнится m[6], на втором – m[5], на третьем – m[4], … , на i-м шаге – m[l-i].

Для извлечения очередной цифры можно делить число *a* на 10 с получением целочисленного остатка, который (остаток) и записывается как очередной элемент массива. Для следующего шага число *a* целочисленно делится на 10, чтобы следующим разряд числа *a*, в свою очередь, оказался справа:

i	m[l-i]= *a* «получение остатка от деления» на 10	a=a«целочисленное деление» на 10
0	m[6-0]=745319 mod 10=9; m[6]=9	*a*=745319 div 10=74531
1	m[6-1]=74531 mod 10=1; m[5]=1	*a*=74531 div 10=7453
2	m[6-2]=7453 mod 10=3; m[4]=3	*a*=7453 div 10=745
3	m[6-3]=745 mod 10=5; m[3]=5	*a*=745 div 10=74
4	m[6-4]=74 mod 10=4; m[2]=4	*a*=74 div 10=7
5	m[6-5]=7 mod 10=7; m[1]=7	*a*=7 div 10=0

Блок-схема алгоритма с формированием массива:



Далее приводится программная реализация алгоритма. При ее изучении стоит обратить внимание на способы описания массива в различных языках:

- C# позволяет описывать переменные и массивы в любом месте кода до первого использования. Поэтому сначала вводится переменная *a*, потом определяется ее длина, а потом – командой int[] m = new int[l + 1] - создается массив требуемого размера. Обращаем внимание, что размер массива указан как l+1, а не l. Причина тому – нумерация массива с нуля: при объявлении массива размером, например, в 3 элемента, будет созданы три элемента: m[0], m[1] и m[2]. То есть элемент m[3] существовать не будет. Для совпадения классической нумерации символов-цифр в числе с номерами элементов массива можно создать массив на единицу больший. Если этого не сделать, при дельнейшей программной реализации надо помнить о смещении номеров; например, ответ будет формироваться как m[b-1]+m[c-1].
- Pascal ABC предполагает, что описание всех переменных и массивов производится до программного кода. Но, так как длина массива m заранее неизвестна (вводимое число может быть любой

длины), то создается так называемый динамический массив – размер которого в процессе работы программы может меняться. При описании его размер не указывается; после ввода *a* и определения количества цифр процедурой SetLength уточняется размер массива и количества отводимой под него памяти.

- в VBA, несмотря на то, что теоретически описание переменных и массивов может размещено в любом месте кода до первого использования, также, как и в Pascal ABC, требуется константа, определяющая размер массива. А потому, после ввода *a*, и определения ее длины l, команда Dim m(l) as Integer недопустима: l – переменная, а не константа. Следовательно, в VBA, подобно PascalABC, сначала описывается динамический массив без указания его длины, а потом – командой ReDim m(l) уточняется его размер.

C# с формированием массива:

```
static void Main(string[] args)
        {
            Console.WriteLine("введите первый параметр");
            int a = int.Parse(Console.ReadLine());
            string s = Convert.ToString(a);
            int l = s.Length;
            int b, c;
                Console.WriteLine("введите второй параметр");
                b = int.Parse(Console.ReadLine());
                Console.WriteLine("введите третий параметр");
                c = int.Parse(Console.ReadLine());
                int i;
                int[] m = new int[l + 1];          //описание массива
                for (i = 0; i < l; i++)
                {
                    m[l-i] = a - (a / 10) * 10;
                    a = a / 10;
                }
            Console.WriteLine("{0}", m[b] + m[c]);
        }
```

Pascal ABC с формированием массива:

```
var a,b,c,l,i:integer;
s:string;
m:array of integer;        //описание массива
begin
      writeln('введите первый параметр');
      readln(a);
      str(a,s);
      l:=length(s);
      SetLength(m,l+1);  //выделение памяти под заявленный массив
writeln('введите второй параметр');
readln(b);
writeln('введите третий параметр');
```

```
readln(c);
      for i:=0 to l-1 do
      begin
      m[l - i]:= a Mod 10;
      a:= a div 10;
      end;
writeln(m[b]+m[c]);
end.
```

VBA с формированием массива:

```
Sub avtomatm()
Dim a As Long
Dim l As Integer
Dim b As Integer
Dim c As Integer
Dim i As Integer
Dim m() As Integer              'описание массива
      a = InputBox("Введите первый параметр")
      l = Len(CStr(a))
      ReDim m(l)                'выделение памяти под заявленный массив
      b = InputBox("Введите второй параметр")
      c = InputBox("Введите третий параметр")
For i = 0 To l - 1
m(l - i) = a Mod 10
a = a \ 10
Next i
      MsgBox (m(b) + m(c))
End Sub
```

Подход к решению задачи с формированием массива реализуется кодом меньшей длины, чем предыдущие, также позволяет избежать создания переменной *a*1 и двойного прохода цикла для получения двух слагаемых, но требует выделения памяти под массив m.

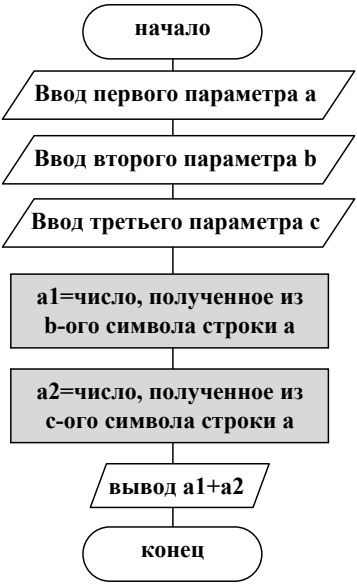
Альтернативным способом решения задачи является ввод параметра *a* как строковой переменной.

Как видно из приведенной ниже блок-схемы, данный вариант алгоритма самый лаконичный из приведенных. Но в рамках реализации данного алгоритма, требуется решить две задачи:

1. извлечь символы из b-ой и из c-ой позиции;
2. преобразовать их в числа.

Задачи извлечения символов и преобразования их в числа решаются следующим образом:

в C# извлечение подстроки из строки производится при помощи метода a.Substring(p, k), где a – строка, из которой надо извлечь подстроку, p – номер символа в строке a, начиная с которого надо извлекать, k – количество символов в извлекаемой подстроке. Обращаем внимание, что в нижележащем коде p=b-1 по причине того, что нумерация символов в строке начинается с нуля.



- Преобразование строки в целое число *a*1 производится с помощью *a*1= int.Parse(s), где s – строка, из которой надо получить соответствующее ей число.
- в Pascal ABC извлечение подстроки из строки производится при помощи функции copy(a,p,k), где a – строка, из которой надо извлечь подстроку, p – номер символа в строке a, начиная с которого надо извлекать, k – количество символов в извлекаемой подстроке.

Преобразование строки s в целое число *a*1 производится с помощью процедуры val(s,a1,err), где err – код результата преобразования: если преобразование успешно, то err=0, иначе err>0.

- в VBA извлечение подстроки из строки производится при помощи функции Mid(a, p, k), где a – строка, из которой надо извлечь подстроку, p – номер символа в строке a, начиная с которого надо извлекать, k – количество символов в извлекаемой подстроке.

Преобразование строки s в целое число *a*1 производится с помощью *a*1=CInt(s).

C#, первый параметр – строка:

```
static void Main(string[] args)
        {
            Console.WriteLine("введите первый параметр");
            string a = Console.ReadLine();
            int b, c;
```

```
            Console.WriteLine("введите второй параметр");
            b = int.Parse(Console.ReadLine());
            Console.WriteLine("введите третий параметр");
            c = int.Parse(Console.ReadLine());
            int a1 = int.Parse(a.Substring(b - 1, 1));
            int a2 = int.Parse(a.Substring(c - 1, 1));
            Console.WriteLine("{0}", a1 + a2);
        }
```

Pascal ABC, первый параметр – строка:

```
var b,c,a1,a2,e:integer;
a:string;
begin
writeln('введите первый параметр');
readln(a);
writeln('введите второй параметр');
readln(b);
writeln('введите третий параметр');
readln(c);
val(copy(a,b,1),a1,e);
val(copy(a,c,1),a2,e);
writeln(a1+a2);
end.
```

VBA, первый параметр – строка:

```
Sub avtomat()
Dim a As String
Dim b As Integer
Dim c As Integer
Dim a1 As Integer
Dim a2 As Integer
     a = InputBox("'введите первый параметр ")
     b = InputBox("введите второй параметр")
     c = InputBox("введите третий параметр")
a1 = CInt(Mid(a, b, 1))
a2 = CInt(Mid(a, c, 1))
     MsgBox (a1 + a2)
End Sub
```

Как видно из приведенных кодов, код решения задачи с представлением первого параметра в строковом формате во всех трех языках выгодно отличается от предыдущих кодов, как по размеру кода, так и по и сложности.

Однако на текущий момент, задача не является полностью решенной: для простоты объяснения в алгоритме был опущен контроль корректности вводимых данных. Согласно входным данным задачи, при слишком больших значениях второго и третьего параметра должно выводиться сообщение об ошибке.

Если протестировать любой из приведенных кодов с слишком большими значениями второго и третьего параметров, то большая часть кодов не будет корректно завершена, а будет прервана с выводом сообщения об ошибке. Аналогичная ситуация будет наблюдаться при некорректном ввода первого параметра (например, «98в78»).

Для того, чтобы обработать некорректные наборы входных данных, при которых программный код не может быть корректно завершен, можно воспользоваться так называемыми операторами исключений. Данные операторы при возникновении ошибки не прерывают ход программы с сообщением об ошибке, но переходят на блок команд, который предназначен для обработки ошибок.

Рассмотрим операторы исключений на примере их применения для перехвата ошибок.

В C# для этих целей используется оператор try catch. Защищаемый код размещается внутри блока try. Обратите внимание: в приведенном примере весь исполняемый код программы (реализованный по предыдущему алгоритму, когда параметр a является строкой) внесен в блок try.

Блок catch содержит блок, который обрабатывает ошибочные ситуации, происходящие в коде блока try. В задаче требуется только напечатать на экране текст «error». Но в приведенном примере, помимо печати текстового сообщения, в блоке catch производится повторный вызов самого метода Main. Таким образом, по факту любой ошибки, на экран будет выведен текст «error» и повторно запущен метод Main для повторного ввода входных параметров.

C# с сообщением «error»:

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("введите первый параметр");
        string a = Console.ReadLine();
        int b, c;
        Console.WriteLine("введите второй параметр");
        b = int.Parse(Console.ReadLine());
        Console.WriteLine("введите третий параметр");
        c = int.Parse(Console.ReadLine());
        int a1 = int.Parse(a.Substring(b - 1, 1));
        int a2 = int.Parse(a.Substring(c - 1, 1));
        Console.WriteLine("{0}", a1 + a2);
    }
    catch
        {
            Console.WriteLine("error");
            Main(args );
        }
}
```

В VBA для аналогичных целей можно использовать оператор OnError GoTo <метка>. При его применении при любой ошибке выполнения кода программа переходит на указанную метку, в примере – на метку m_err.

Метка m_err содержит команду печати «error» и повторный вызов макроса avtomat.

При отсутствии ошибки по успешному окончанию выполнению всех команд кода записана команда безусловного перехода GoTo m_end, которая «обходит» обработку ошибки, передая управление на конец макроса.

VBA с сообщением «error»:

```
Sub avtomat()
Dim a As String
Dim b As Integer
Dim c As Integer
Dim a1 As Integer
Dim a2 As Integer
On Error GoTo m_err
    a = InputBox("введите первый параметр")
    b = InputBox("введите второй параметр ")
    c = InputBox("введите третий параметр ")
a1 = CInt(Mid(a, b, 1))
a2 = CInt(Mid(a, c, 1))
    MsgBox (a1 + a2)
    GoTo m_end
m_err: MsgBox ("error")
avtomat
m_end: End Sub
```

Аналогичный алгоритм (параметр a как строка) в PascalABC ошибки не дает в силу специфики работы процедуры val, преобразующей строку в число: у этой процедуры, как отмечено выше, третий параметр является индикатором ошибки: если он равен нулю – преобразование прошло успешно; если он больше нуля, то в процессе преобразования были ошибки. Таким образом, для получения сообщения «error» можно воспользоваться значением данного параметра:

```
var b,c,a1,a2,e1,e2:integer;
a:string;
label m1;
begin
m1:writeln('введите первый параметр');
readln(a);
writeln('введите второй параметр');
readln(b);
writeln('введите третий параметр');
readln(c);
val(copy(a,b,1),a1,e1);
val(copy(a,c,1),a2,e2);
if (e1>0) or (e2>0) then
begin
writeln('error');
goto m1;
end
else
writeln(a1+a2);
end.
```

В проведенном коде проверяется код результата преобразования для обоих символов. В случае ошибки – о ней сообщается и оператором goto происходит переход на метку m1 – на начало работы программы.

Однако оператор, аналогичный try catch в C#, в Pascal ABC также существует. Рассмотрим оператор обработки исключений try except на примере кода с формированием массива.

Блок, распложенный после try является защищаемым блоком. Если при выполнении программы в нем происходит ошибка, то он завершается и выполнение передается блоку except.

Блок except запускается только при ошибке; при отсутствии ошибки данный блок игнорируется и не выполняется.

Pascal ABC с формированием массива и с сообщением «error» try except:

```
var a,b,c,l,i:integer;
s:string;
m:array of integer;       //описание массива
begin
try
      writeln('введите первый параметр');
      readln(a);
      str(a,s);
      l:=length(s);
      SetLength(m,l+1);
      writeln('введите второй параметр');
      readln(b);
      writeln('введите третий параметр');
      readln(c);
      for i:=0 to l-1 do
      begin
      m[l - i]:= a Mod 10;
      a:= a div 10;
      end;
      writeln(m[b]+m[c]);
except
writeln('error');
end;
end.
```

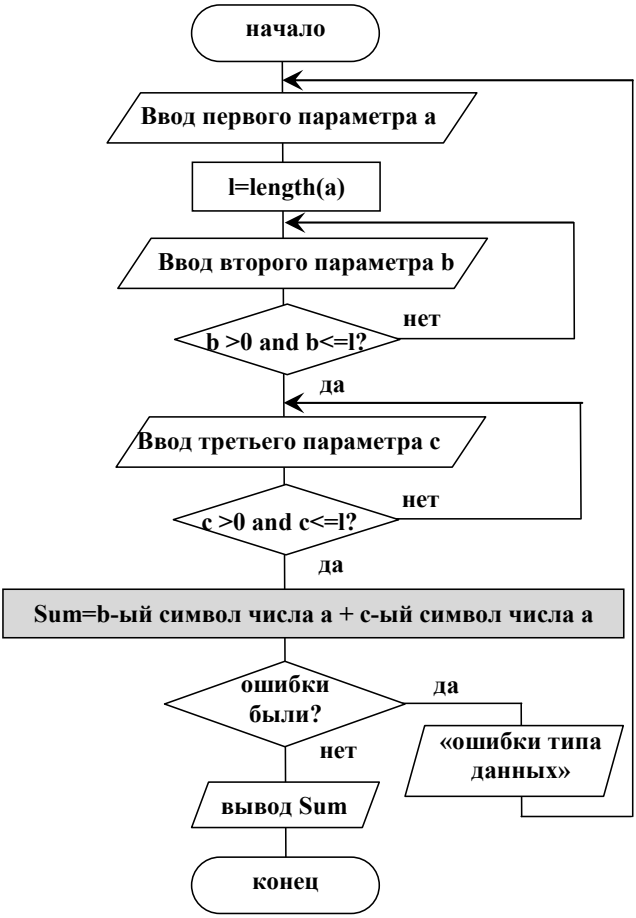
На этом исходную задачу можно считать решенной. Однако, несколько отступая от исходных условий задачи (при некорректных входных данных автомат должен выдать ошибку) полезно также рассмотреть вариант алгоритма, который не только выводит сообщение «error», но и предлагает пользователю повторно ввести верное значение некорректно-введенного параметра.

Учтем, что ошибки параметров могут иметь различную природу:

- ошибка типа данных: ввести вместо цифр числа строковые символы;
- ошибка числового значения: ввести второй или третий параметр, значение которого превышает количество символов в первом параметре.

Первый тип ошибок будем обрабатывать уже описанным образом с помощью операторов обработки исключений.

Во избежание второго типа ошибок будем анализировать вводимое значение; при несоответствии длине первого параметра будет предлагаться ввести значение повторно.



C# контроль вводимых параметров:

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("введите первый параметр");
        string a = Console.ReadLine();
```

```
            int b, c;
            int l = a.Length;
            do
            {
            Console.WriteLine("введите второй параметр");
            b = int.Parse(Console.ReadLine());
            } while ((b > l)||(b<1));
            do
            {
            Console.WriteLine("введите третий параметр");
            c = int.Parse(Console.ReadLine());
            } while ((c > l) || (c < 1));
            int a1 = int.Parse(a.Substring(b - 1, 1));
            int a2 = int.Parse(a.Substring(c - 1, 1));
            Console.WriteLine("{0}", a1 + a2);
        }
        catch
            {
               Console.WriteLine("ошибка типа данных");
               Main(args );
            }
    }
```

Pascal ABC контроль вводимых параметров:

```
var b,c,l,a1,a2,e1,e2:integer;
a:string;
begin
      try
      writeln('введите первый параметр');
      readln(a);
      l:=length(a);
      repeat
      writeln('введите второй параметр');
      readln(b);
      until((b <= l)and(b>0));
      repeat
      writeln('введите третий параметр');
      readln(c);
      until((c <= l)and(c>0));
      val(copy(a,b,1),a1,e1);
      val(copy(a,c,1),a2,e2);
      if (e1=0) and(e2=0) then writeln(a1+a2)
      else writeln('ошибка типа данных');
            except
            writeln('ошибка типа данных');
            end;
end.
```

VBA контроль вводимых параметров:

VBA контроль вводимых параметров:

```
Sub avtomat()
Dim a As String
Dim b As Integer
Dim c As Integer
Dim a1 As Integer
Dim a2 As Integer
```

```
Dim l As Integer
On Error GoTo m_err
    a = InputBox("Введите первый параметр")
    l = Len(CStr(a))
    Do
    b = InputBox("Введите второй параметр")
    Loop While b > l Or b < 1
    Do
    c = InputBox("Введите третий параметр")
    Loop While c > l Or c < 1
a1 = CInt(Mid(a, b, 1))
a2 = CInt(Mid(a, c, 1))
    MsgBox (a1 + a2)
    GoTo m_end
m_err: MsgBox ("ошибка типа данных")
avtomat
m_end: End Sub
```

Приведенные коды дополнены расчетом l – количества символов в первом параметре и двумя практически одинаковыми циклами по условию: циклы будут повторяться, если введенное значение второго/ третьего параметра меньше единицы или больше l. Условие проверяется на выходе из цикла; при корректном вводе параметра цикл выполняется один раз.

Недостатком кода является появление в нем повторяющихся, почти идентичных, блоков – для ввода и контроля второго и третьего параметров; отличие – только в имени вводимого параметра. Для двух параметров данный недостаток не столь существенен и вполне допустим. Но, если предположить большее количество одинаково вводимых параметров, то надо отметить, что стоит избегать в коде большого количества практически одинаковых блоков кода, так как:

1. длинный код рассеивает внимание и затрудняет восприятие;
2. вероятность ошибки невнимательности существенно возрастает, а ее обнаружение затруднено по первой причине;
3. при необходимости внести изменения, их придется вносить во все блоки, что увеличивает трудоемкость и повышает вероятность ошибки.

Исправить этот недостаток можно, применяя подпрограммы (методы, процедуры, функции, и прочее).

Подпрограмма в общем смысле слова – некий самостоятельно выполняемый фрагмент кода. При ее вызове ей можно передать какие-либо параметры для выполнения ее работы. По завершении подпрограмма может вернуть какие-либо результаты.

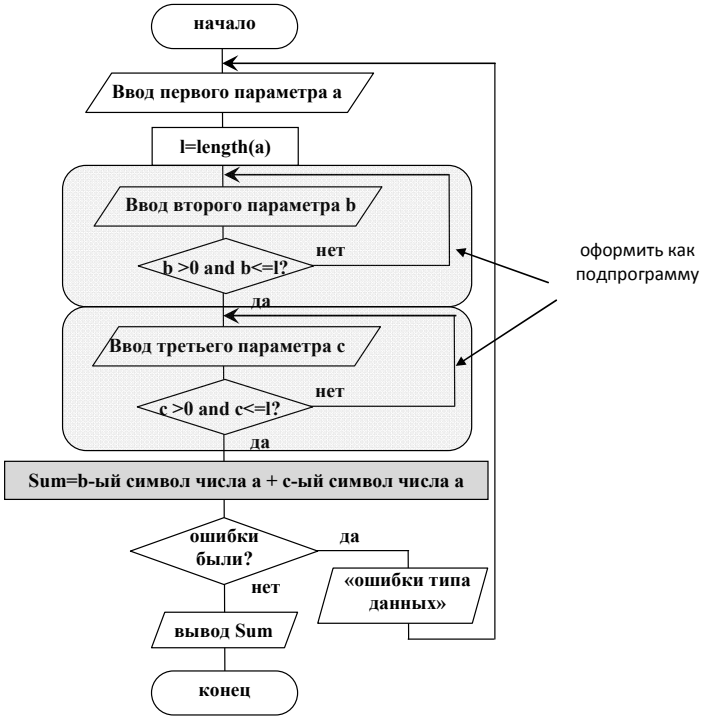
Создание исполняемого кода из отдельных фрагментов поддерживается практически во всех языках программирования, но реализовано по-разному.

В C# фрагмент кода носит название «метод». При запуске первым выполняется метод Main. Из метода Main могут быть вызваны прочие методы (если они существуют). Методы могут принимать параметры, могут работать без параметров, могут по завершении работы передавать обратно результат методу, который их вызвал, могут – не передавать.

В Pascal ABC подпрограммы подразделяются на процедуры и функции. И процедура, и функция представляют собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей основной программы. Но функция, в отличие от процедуры, в результате своего выполнения возвращает значение, которое может быть использовано в выражении в вызываемом модуле.

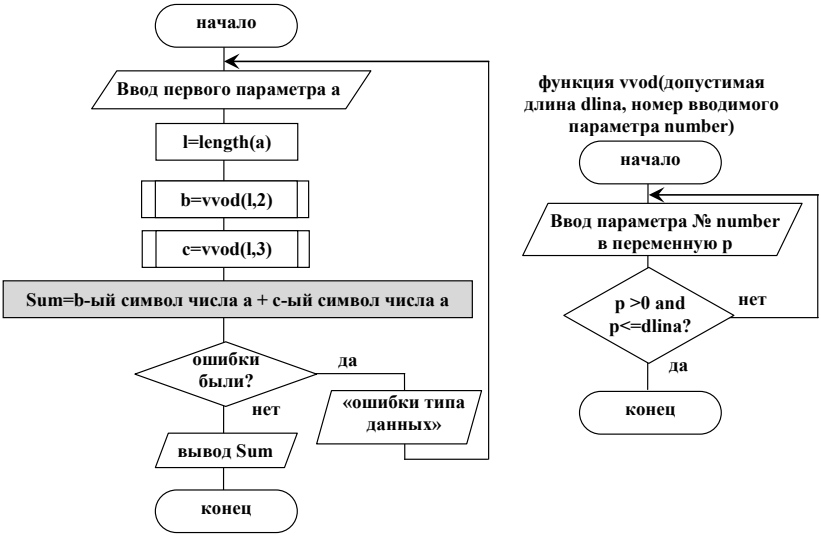
В VBA исполняемые модули носят название «процедура» (или «макрос») и «функция». Аналогично Pascal ABC функция возвращает какое-либо значение. А процедура может либо не возвращать никаких значений, либо наоборот – возвращать несколько значений.

Возвращаясь к коду, нам надо изменить алгоритм так, чтобы повторяющийся фрагмент алгоритма оформить как отдельную подпрограмму:



Подпрограмма должна обеспечивать ввод и контроль параметра; если значение параметра не корректно, предложить ввести параметр заново. Для этого она получать от основной программы количество символов в первом параметре (как ограничение величины параметра). Так же ей нужно получать от основной программы номер вводимого параметра (первый раз – это второй параметр, потом – третий). Подпрограмма должна возвращать число – значение успешно-введенного параметра. Следовательно, логично подпрограмму оформить как функцию.

Блок-схема варианта с подпрограммой приведена ниже:



Функция vvod вызывается первый раз с параметрами 1 (количество символов в первом параметре) и 2 (требуется ввести параметр номер 2). Результат ее работы записывается в переменную *b*. Второй вызов функции vvod производится также с параметрами 1 (количество символов в первом параметре) и 3 (требуется ввести параметр номер 3). Результат ее работы записывается в переменную *c*.

В C# понятия «функция» не существует; в C# для этого создается статический метод vvod, который возвращает целочисленное значение. Указание на тип возвращаемого значения приведено в заголовке метода: static **int** vvod(int dlina,int number). Также из кода видно, что возвращаемое значение *p* (return p) описано как целое число (int p).

В заголовке же указываются передаваемые методу величины и их тип: static int vvod(**int dlina**,**int number**). При вызове метода *b* =

vvod(l, 2) из основного метода Main значение переменной l записывается в переменную dlina; число 2- в переменную number.

C# контроль вводимых параметров с методом vvod ввода второго и третьего параметров:

```
class Program
{
    static int vvod(int dlina,int number)
    {
        int p;
        do
            {
            Console.WriteLine("введите параметр № {0}",number);
            p = int.Parse(Console.ReadLine());
            } while ((p > dlina)||(p<1));
        return p;
    }
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("введите первый параметр");
            string a = Console.ReadLine();
            int b, c;
            int l = a.Length;
            b = vvod(l, 2);
            c = vvod(l, 3);
            int a1 = int.Parse(a.Substring(b - 1, 1));
            int a2 = int.Parse(a.Substring(c - 1, 1));
            Console.WriteLine("{0}", a1 + a2);
        }
        catch
            {
               Console.WriteLine("ошибка типа данных");
               Main(args );
            }
    }
}
```

В Pascal ABC тип функции vvod (то есть тип возвращаемого функцией значения) описывается также в заголовке: function vvod(dlina:integer; number:integer):**integer**. Также из кода функции видно, что возвращаемое значение *p* (vvod:=p) описано как целое число (var p:integer).

В заголовке же указываются передаваемые функции величины и их тип: function vvod(**dlina:integer**; **number:integer**):integer. При вызове функции *b* = vvod(l, 2) из программы значение переменной l записывается в переменную dlina; число 2- в переменную number.

Pascal ABC контроль вводимых параметров с функцией vvod ввода второго и третьего параметров:

```
function vvod(dlina:integer; number:integer):integer;
var p:integer;
begin
repeat
writeln('введите параметр №', number);
readln(p);
until((p <= dlina)and(p>0));
vvod:=p;
end;

var b,c,l,a1,a2,e1,e2:integer;
a:string;
begin
try
writeln('введите первый параметр');
readln(a);
l:=length(a);
b:=vvod(l,2);
c:=vvod(l,3);
val(copy(a,b,1),a1,e1);
val(copy(a,c,1),a2,e2);
if (e1=0) and(e2=0) then writeln(a1+a2)
else writeln('ошибка типа данных');
except
writeln('ошибка типа данных');
end;
end.
```

Реализация VBA сходна с реализацией PAscalABC. Тип функции vvod (то есть тип возвращаемого функцией значения) описывается также в заголовке: Function vvod(dlina As Integer, number As Integer) **As Integer**).

В заголовке же указываются передаваемые функции величины и их тип: Function vvod(**dlina As Integer**, **number As Integer**) As Integer. При вызове функции *b* = vvod(l, 2) из процедуры avtomat значение переменной l записывается в переменную dlina; число 2- в переменную number.

VBA контроль вводимых параметров с функцией vvod ввода второго и третьего параметров:

```
Function vvod(dlina As Integer, number As Integer) As Integer
    Do
    vvod = InputBox("введите параметр № " & number)
    Loop While vvod > dlina Or vvod < 1
End Function

Sub avtomat()
Dim a As String
Dim b As Integer
Dim c As Integer
Dim a1 As Integer
Dim a2 As Integer
Dim l As Integer
```

```
On Error GoTo m_err
    a = InputBox("Введите первый параметр")
    l = Len(CStr(a))
    b = vvod(l, 2)
    c = vvod(l, 3)
a1 = CInt(Mid(a, b, 1))
a2 = CInt(Mid(a, c, 1))
    MsgBox (a1 + a2)
    GoTo m_end
m_err: MsgBox ("Ошибка типа данных")
avtomat
m_end: End Sub
```

Анализ количества кода показывает что, при использовании подпрограмм, общий размер кода данной задачи существенно не изменился, в некоторых вариантах кода имеет место незначительно увеличение (на пару строк), но размер основного исполняемого кода уменьшился, и сам код воспринимается проще: внимание не рассеивается на детальное описание ввода каждого параметра.

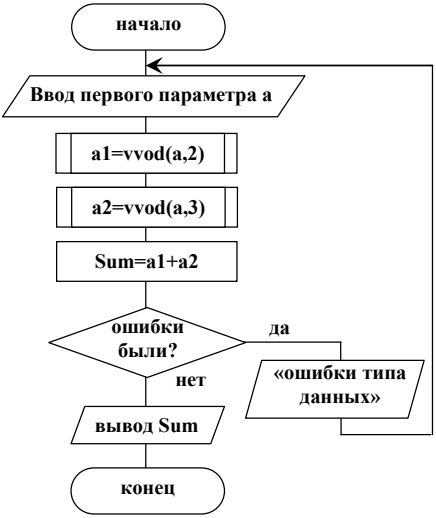
Общее, крайне незначительное, увеличение кода в данном случае объясняется простотой программы и наличием всего двух параметров: в более сложных и длинных кодах и при более частом вызове подпрограмм выигрыш от их применения в размере кода ощутим.

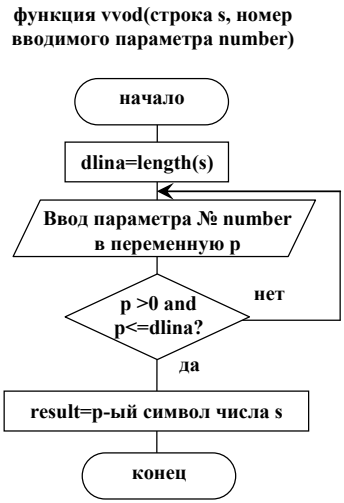
Анализ переменных кода показывает, что переменная l не используется в основном коде. Ее вычисляют только для того, чтобы передать подпрограмме для ввода параметра. Если по логике дальнейшей реализации переменная l не будет задействована в основном коде, то блок-схему можно изменить, перенеся вычисление l в подпрограмму. При этом для вычисления l подпрограмме необходимо передать сам первый параметр *a*.

Если подпрограмма получает значение введенной строки (параметр *a*), то на эту же подпрограмму можно возложить извлечение требуемого символа из этой строки и возврат в основную программу не номера символа, а цифры с требуемой позиции, то есть подпрограмма будет возвращать слагаемое.

Подпрограмма vvod в данном случае будет получать строкой параметр *s* и целочисленный параметр number; тип возвращаемого значения – целое число.

Обратите внимание, что при подобном подходе основной код программы существенно уменьшает свой размер и количество используемых переменных. А размер кода, количество переменных и сложность подпрограммы увеличивается.





C# с методом vvod для ввода, контроля вводимого параметра и извлечения нужного числа из первого параметра:

```
class Program
{
    static int vvod(string s, int number)
    {
        int dlina = s.Length;
        int p;
        do
            {
            Console.WriteLine("введите параметр № {0}",number);
            p = int.Parse(Console.ReadLine());
            } while ((p > dlina)||(p<1));
        int result = int.Parse(s.Substring(p - 1, 1));
        return result;
    }
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("введите первый параметр");
            string a = Console.ReadLine();
            int a1 = vvod(a, 2);
            int a2 = vvod(a, 3);
            Console.WriteLine("{0}", a1 + a2);
        }
        catch
            {
               Console.WriteLine("ошибка типа данных");
               Main(args );
            }
    }
}
```

Реализация в Pascal ABC по причине внутренней логики Pascal ABC несколько отличается от прочих. Преобразование типа реализуется не функцией, а процедурой val, которая при ошибке преобразования не приводит к аварийному завершению кода в целом, а позволяет продолжить работу программы, сигнализируя о наличии ошибке преобразования параметром err, отличном от нуля.

Таким образом, предложенный код также реализует не функцию vvod, а процедуру vvod, которая в отличии от функции не только возвращает результат работы, но параметр err, который является индикатором ошибки: если он равен нулю – ошибки не было; если больше нуля – есть ошибка преобразования.

Pascal ABC с процедурой vvod для ввода, контроля вводимого параметра и извлечения нужного числа из первого параметра:

```
procedure vvod(s:string; number:integer; Var res,err:integer);
var p:integer;
begin
try
repeat
writeln('введите параметр №', number);
readln(p);
until((p <= length(s))and(p>0));
val(copy(s,p,1),res,err);
except
err:=1;
end;
end;

var a1,a2,e1,e2:integer;
a:string;
begin
writeln('введите первый параметр');
readln(a);
vvod(a,2,a1,e1);
vvod(a,3,a2,e2);
if (e1>0)or (e2>0) then writeln('ошибка типа данных')
else
writeln(a1+a2);
end.
```

VBA с функцией vvod для ввода, контроля вводимого параметра и извлечения нужного числа из первого параметра:

```
Function vvod(s As String, number As Integer) As Integer
    Dim dlina As Integer
    Dim p As Integer
    dlina = Len(CStr(s))
    Do
    p = InputBox("Введите параметр №" & number)
    Loop While p > dlina Or p < 1
    vvod = CInt(Mid(s, p, 1))
End Function
```

```
Sub avtomat()
Dim a As String
Dim a1 As Integer
Dim a2 As Integer
On Error GoTo m_err
    a = InputBox("введите первый параметр")
   a1 = vvod(a, 2)
   a2 = vvod(a, 3)
    MsgBox (a1 + a2)
    GoTo m_end
m_err: MsgBox ("ошибка типа данных")
avtomat
m_end: End Sub
```

При выборе предпочтительного варианта решения из двух последних стоит руководствоваться следующими соображениями:

- при модульном подходе разбиение на отдельные модули выполняется с тем расчетом, чтобы каждый модуль полностью выполнял свою задачу;
- задачи модулей не должны дублироваться;
- модули должны поддерживать максимальную гибкость.

Как видно, из приведенных кодов, вариантов решения данной задачи большое количество. Удачность реализации во многом определяется языком программирования и его возможностями. Любой из работоспособных кодов засчитывается как правильный ответ.

Использование методов (функций, процедур) при решении задач на олимпиаде не является обязательным, но наилучшим образом характеризует исполнителя работы.

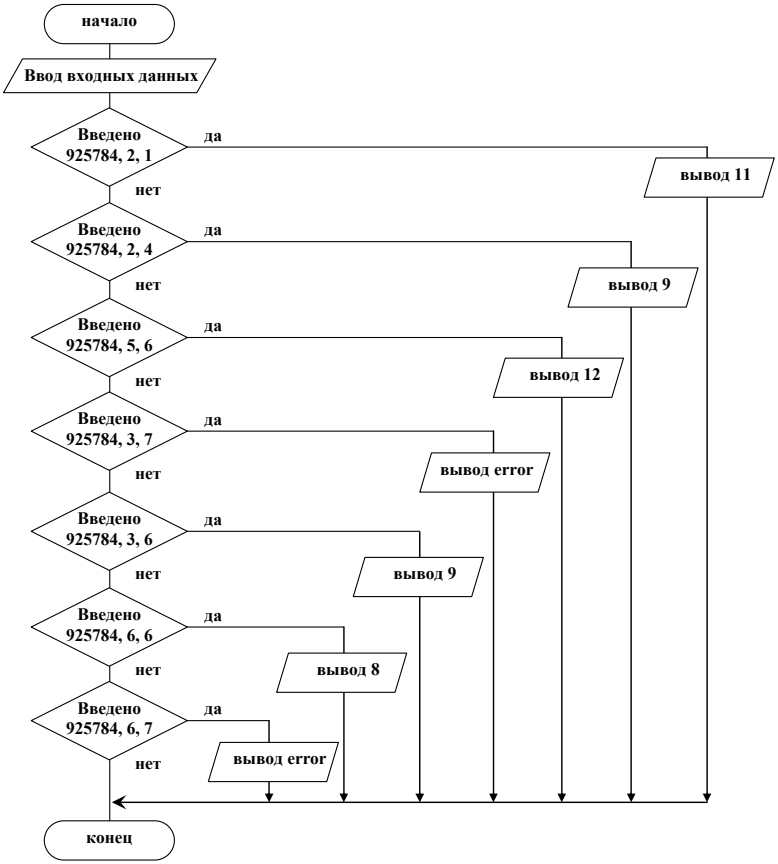
Типичные ошибки, допущенные при реализации задачи

1. Непонимание логики работы автомата, неумение увидеть закономерности.

2. Решение в частном виде программированием заданных выходных данным по заданным входным. Например, нижеприведенная блок-схема не является правильным ответом.

3. Ответ «Сумма указанных элементов» является недостаточным, так как данная задача предлагает продемонстрировать умение программировать.

4. Отсутствие описания переменных: приводится код программы, соответствующий логике работы алгоритма, но без описания переменных. В этом случае снижаются баллы за небрежное написание программы: даже в тех языках программирования, которые не требуют обязательного объявления переменных, рекомендуется описывать переменные для экономии памяти и во избежание ряда ошибок.



5. Неумение (нежелание) представлять исходные данные в строковом формате. Само по себе это не является ошибкой, но, как показано при разборе вариантов решения задачи, в некоторых случаях именно этот подход обеспечивает простоту решения задачи.

Задача 6 (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 деся-

тичных 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
```
readln(s); readln(n);
```

C#
```
string s = Console.ReadLine();
int n=int.Parse(Console.ReadLine());
```

VBA
```
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 из 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 ко-

торому 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();
}
```

```
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 «–» .

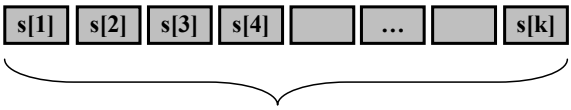
При формировании расстановок надо помнить, что необязательно между двумя цифрами должен стоять символ «+» или «–». Наилучший способ разрешить задачу расстановки символов «+», «–» или отсутствия символа – создать строковый массив ins из трех элементов «+», «–», «» (пустая строка) и вставить элементы массива 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] = "-";
```

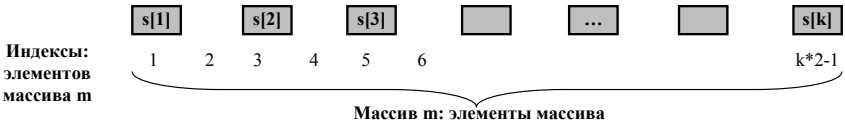
VBA
```
Dim ins(3) As
String
ins(1) = ""
ins(2) = "+"
ins(3) = "-"
```

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

Размерность 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.Substring(i,1);
…
```

C#

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

VBA

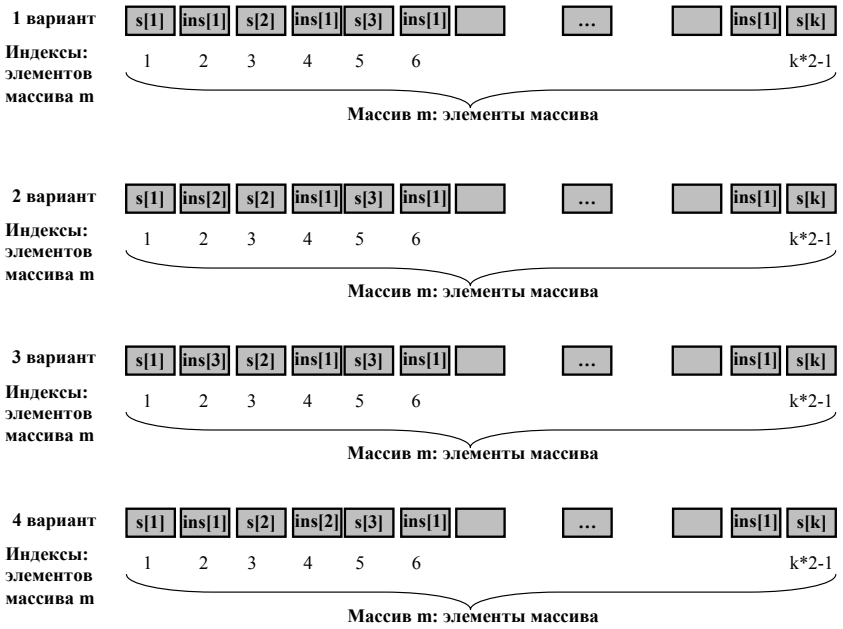
```
k = Len(s)
Dim m() As
String
ReDim m(k * 2)

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

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

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

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

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

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

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

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];`

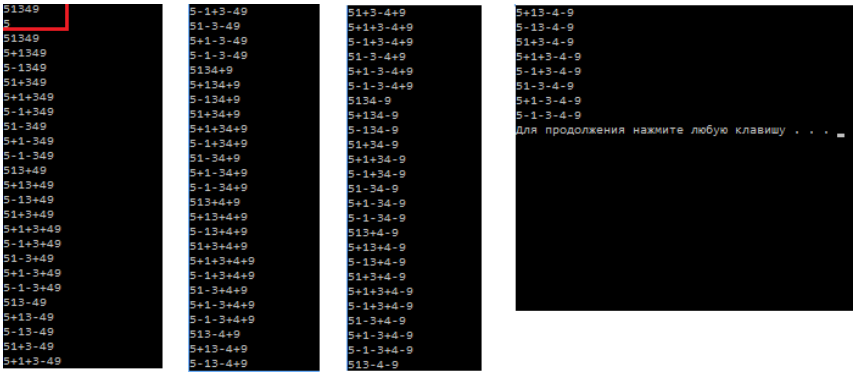
VBA	`For q = 1 To 3 ^ (k - 1)`
	`    For j = 2 To k * 2 - 1 Step 2`
	`    m(j) = ins(1 + (q / 3 ^ ((j - 2) / 2)) Mod 3)`
	`    Next j`
	`Next q`

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.

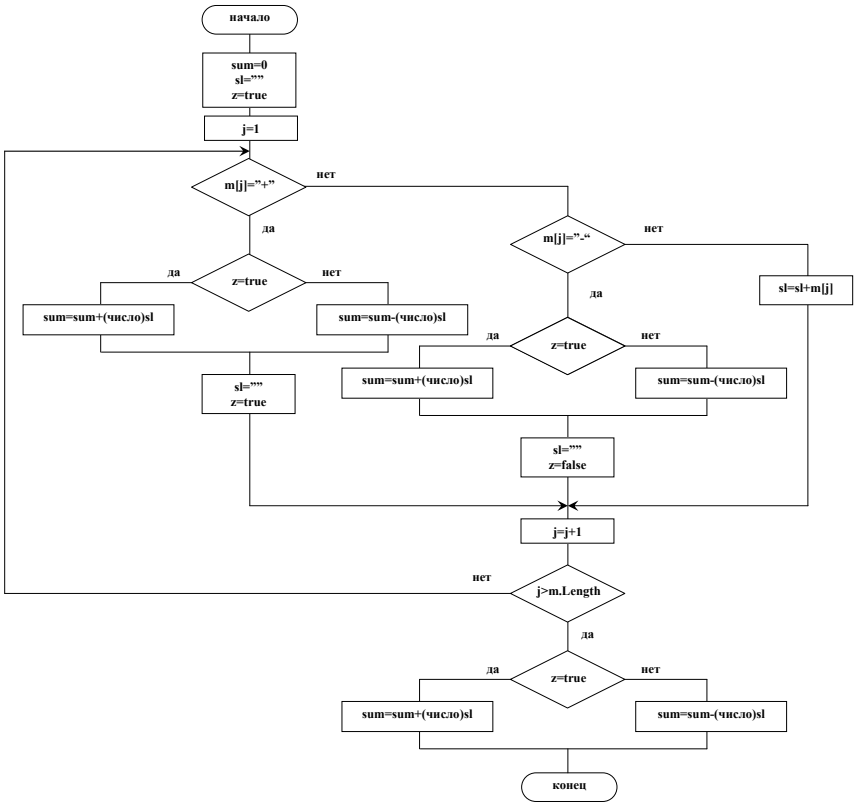
2. Произвести математические расчеты (сложение или вычисления в зависимости от знака).

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.

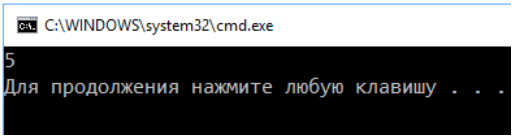
В приведенной блок-схеме остался необъясненным один момент: переменная *sl* – строкового типа, а по логике задачи требуется числовое значение. Для решения данного вопроса воспользуемся преобразованием типов данных.

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:

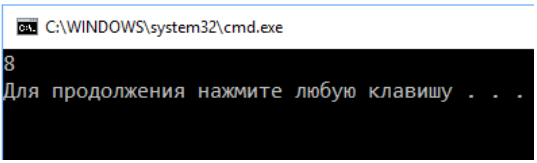
```
static void Main(string[] args)
{
    int i = 5;
    double d = 8.8;
    d = i;
    Console.WriteLine(d);
}
```



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:

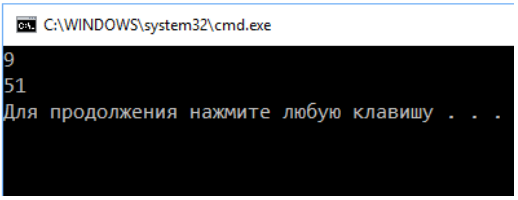
```
static void Main(string[] args)
{
    int i = 5;
    double d = 8.8;
    i=(int)d;
    Console.WriteLine(i);
}
```



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 есть риск, 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 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 хорошо, 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 способ.

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 дополнительно перебирать строки данного массива (увеличение времени выполнения программы).
- Ввести строковую переменную res для хранения лучшего варианта расстановки. Исходное значение res=”” (пустая строка).
- Ввести 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.

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

Подсчитать значение *kolz* в процессе вычисления суммы каждой последовательности. 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
```

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 повторить программную реализацию задач.

Для практического закрепления навыков авторы рекомендуют самостоятельно разработать свои программные коды задач 4, 5 и 6, отличные от кодов, предложенных авторами. В пособии показано, что для любой задачи можно найти различные пути ее решения; эти задачи дают возможность попрактиковаться в реализации логики алгоритма.

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

Методическое пособие по подготовке к олимпиадам школьников инженерной направленности

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

11 класс

В авторской редакции

Подписано в печать 06.12.17	Бумага офсетная	Уч.-изд. л. 4,4
Формат 60 × 90 $^{1}/_{16}$	Печать офсетная	Заказ

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

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

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