№4 Издание основано в 1995 г. inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА

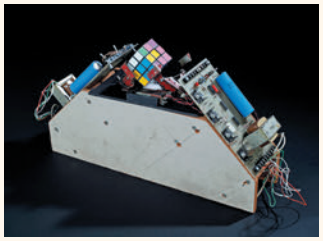
30 апреля — 100 лет со дня рождения Клода Элвуда Шеннона, человека, который придумал бит

электронная ВЕРСИЯ ЖУРНАЛА в Личном кабинете на сайте www.1september.ru

издательский дом 1september.ru
Первое сентября
апрель 2016
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

Клод Элвуд Шеннон был человеком увлекающимся. Он был разработчиком первой промышленной игрушки на радиоуправлении, которая выпускалась в 50-е годы в Японии. Также он разработал устройство, которое могло складывать Кубик Рубика (на фото), миникомпьютер для настольной игры Гекс, который всегда побеждал соперника, механическую мышку, которая могла находить выход из лабиринта.



В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Все подписчики журнала имеют возможность получать электронную версию, которая является полной копией бумажной. Для получения электронной версии:

1) Откройте Личный кабинет на портале “Первое сентября” (www.1september.ru).

2) В разделе “Газеты и журналы / Получение” выберите свой журнал и кликните на кнопку “Я — подписчик бумажной версии”.

3) Появится форма, посредством которой вы сможете отправить нам копию подписной квитанции.

После этого в течение одного рабочего дня будет активирована электронная подписка на весь период действия бумажной. Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ
по каталогу “Почта России”: 79066 — бумажная версия, 12684 — электронная версия

http://inf.1september.ru

Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский (редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 18 000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Генеральный директор: Наум Соловейчик
Главный редактор: Артем Соловейчик
Коммерческая деятельность: Константин Шмарковский (финансовый директор)
Реклама, конференции и техническое обеспечение Издательского дома: Павел Кузнецов
Производство: Станислав Савельев
Административно-хозяйственное обеспечение: Андрей Ушков
Педагогический университет: Валерия Арсланьян (ректор)

Школа цифрового века:
facebook.com/School.of.Digital.Age

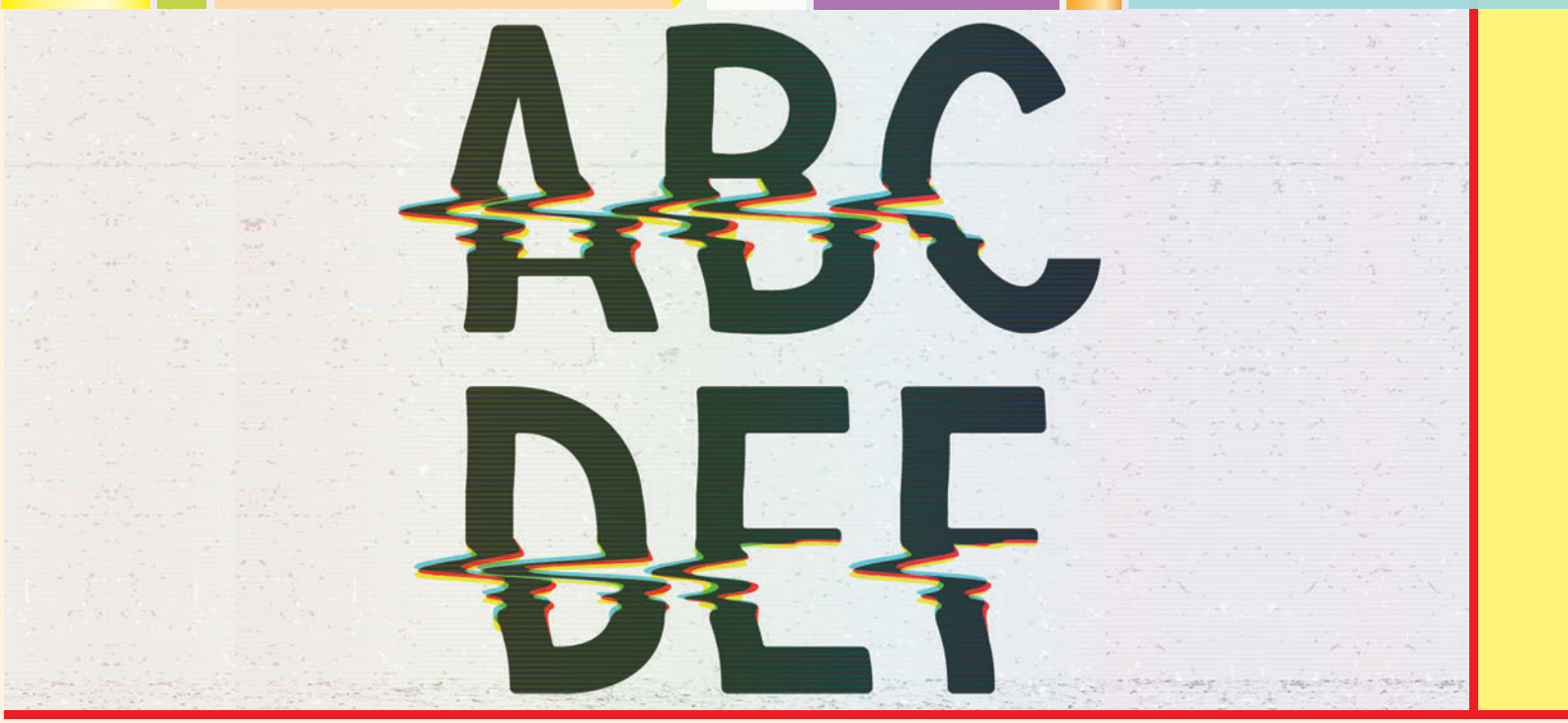
ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА “ПЕРВОЕ СЕНТЯБРЯ”
Английский язык – Е.Богданова
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – и.о. А.Митрофанов
Дошкольное образование – Д.Тюттерин
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – О.Волкова
История – А.Савельев
Классное руководство и воспитание школьников – А.Полякова
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
ОБЖ – А.Митрофанов
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Технология – А.Митрофанов
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школа для родителей – Л.Печатникова
Школьный психолог – М.Чибисова

УЧРЕДИТЕЛЬ:
ООО “ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»”
Зарегистрировано ПИ № ФС77-58447 от 25.06.2014 в Роскомнадзоре
Подписано в печать:
по графику 10.02.2016,
фактически 10.02.2016
Заказ №
Отпечатано в ОАО “Первая Образцовая типография”
Филиал “Чеховский Печатный Двор”
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76

АДРЕС ИЗДАТЕЛЯ:
ул. Киевская, д. 24,
Москва, 121165
Тел./факс: (499) 249-31-38

Отдел рекламы:
(499) 249-98-70
http://1september.ru

ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru



Жонглер, который придумал бит

В 1985 году в английском Брайтоне проходил международный симпозиум по теории информации. В программе симпозиума было много интересных докладов — мир уже вошел в эпоху стремительного развития компьютерных систем и сетей. Но главным лицом представительного собрания ученых стал один-единственный человек. Жонглер. Не самый умелый — он смог управиться лишь с тремя мячиками, хотя его личный рекорд был целых четыре. Именитые ученые выстраивались в длинную очередь, чтобы взять у него автограф. Потому что звали жонглера Клод Элвуд Шеннон. Человек, который придумал бит. Стоящие в очереди говорили, что испытывают такие же чувства, какие испытали бы физики, явись на их конференцию сам сэр Исаак Ньютон.

Карьера Шеннона удивительна. Заложив основы теории информации, введя понятия и сформулировав принципы и теоремы, определившие ключевые инженерные решения компьютерного века, он довольно рано отошел от активной научной деятельности и занялся преподаванием и исследованием увлекающих его совсем не серьезных задач. (Впрочем, Шеннон довольно долго преподавал, среди его учеников был, например, Марвин Мински.)

Шеннон построил несколько жонглирующих машин и даже создал общую теорию жонглирования, которая, впрочем, не помогла ему побить личный рекорд — жонглирование четырьмя мячиками. Еще он испытал свои силы в поэзии, а также разработал разнообразные модели биржи акций и опробовал их (по его словам — успешно) на собственных акциях. В одной из статей Шеннон писал: “Я всегда следовал своим интересам, не думая ни о том, во что они мне обойдутся, ни об их ценности для мира. Я потратил уйму времени на совершенно бесполезные вещи”.

Клод Шеннон родился в 1916 году и вырос в городе Гэйлорде штата Мичиган. Еще в детские годы Клод познакомился как с детальностью технических конструкций, так и с общностью математических принципов. Он постоянно возился с детекторными приемниками и радиоконструкторами, которые приносил ему отец, помощник судьи, и решал математические задачки и головоломки, которыми снабжала его старшая сестра Кэтрин, ставшая впоследствии профессором математики. Клод полюбил эти два мира, столь несхожие между собой,— технику и математику. Позже, в своей диссертации, защищенной в 1940 году, он доказал, что работу переключателей и реле в электрических схемах можно представить посредством алгебры, изобретенной в середине XIX века английским математиком Джорджем Булем. “Просто случилось так, что никто другой не был знаком с этими обеими областями одновременно!” — так скромно Шеннон объяснил причину своего открытия.

30 апреля 2016 года Клоду Элвуду Шеннону исполнилось бы 100 лет.

С.Л. Островский, гл. редактор,
so@1september.ru

Алгоритмизация и программирование

Алгоритмы и исполнители

К.Ю. Поляков,
д. т. н., Санкт-Петербург,
Е.А. Еремин,
к. ф.-м. н., г. Пермь

Ключевые слова:
- алгоритм
- исполнитель
- свойства алгоритма
- способы записи алгоритма
- этапы решения задач на компьютере
- анализ алгоритмов
- бит четности

Формальный исполнитель выполняет команды, входящие в его систему команд исполнителя (СКИ), автоматически, не обдумывая их содержание.

Алгоритм — это точное описание последовательности действий некоторого исполнителя.

Основные свойства алгоритма:

- *дискретность* — алгоритм состоит из отдельных команд, каждая из которых выполняется ограниченное (не бесконечное) время;
- *понятность* — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен;
- *определенность* — при каждом выполнении алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат.

Способы формальной записи алгоритмов

Алгоритмы можно записывать на естественном языке (русском, английском и др.), например, так:

Алгоритм 1. *Даны два натуральных числа. Пока первое число не меньше второго, заменять его на разность первого и второго. Результат работы алгоритма — полученное первое число.*

Словесная запись удобна и привычна, но есть одна проблема: во всех естественных языках есть неоднозначность, поэтому исполнитель-человек может понять алгоритм не так, как задумывал его автор. Особенно трудно разбираться в длинных сло-

весных алгоритмах, занимающих больше десятка строк.

Для того чтобы в алгоритме легче было разобраться, отдельные шаги часто записывают в виде нумерованного списка. Алгоритм 1 может быть записан так:

Вход: два натуральных числа, a и b.

Шаг 1. Если $a < b$, перейти к шагу 4.

Шаг 2. Заменить a на $a - b$.

Шаг 3. Перейти к шагу 1.

Шаг 4. Стоп.

Результат: значение a.

Использовать естественные языки для записи алгоритмов не всегда удобно. Например, если записать алгоритм на русском языке, то люди, не умеющие читать по-русски, не смогут им воспользоваться. В сложных алгоритмах непросто отслеживать переходы между шагами. Поэтому придумали *графическую форму* записи алгоритмов — **блок-схемы**, с которыми вы познакомились в основной школе. Сейчас блок-схемы используются на практике очень редко.

Для того чтобы компьютер мог выполнить алгоритм, нужно записать его на языке программирования — формальном языке, понятном компьютеру.

Любой компьютер (точнее, его процессор) понимает и умеет выполнять только команды в двоичном коде. Но так программы никто не пишет, потому что это очень утомительно, отнимает много времени и приводит к многочисленным ошибкам.

Ближе всего к языку процессора так называемые *языки низкого уровня*, или **языки ассемблера**. В них каждая команда может быть напрямую переведена в соответствующую команду процессора. Это делает специальная программа — *ассемблер* (сборщик).

На языке ассемблера обычно пишут программы, которые должны работать очень быстро и иметь небольшой размер, например, драйверы устройств.

К сожалению, у каждого семейства процессоров свой язык ассемблера, поэтому программы, написанные для одного семейства, не подходят для других. Чтобы решить эту проблему, разработали **языки высокого уровня**, которые построены на основе естественного языка (чаще всего английского). Языки высокого уровня никак не связаны с компьютером, на котором будут выполняться программы. Для каждого типа процессоров (и операционных систем) создается **транслятор** (переводчик) — программа, которая переводит текст программы, написанной на языке программирования, в двоичные коды нужного процессора.

Этапы решения задач на компьютере

В решении любой задачи с помощью компьютера можно выделить несколько этапов.

1. **Постановка задачи**. Нужно четко определить, что в задаче дано (исходные данные) и что нужно найти (результаты).

2. **Формализация**. На этом этапе строится модель задачи, которая записывается с помощью формального языка, например, в виде математических формул или уравнения, связывающего исходные данные и результаты.

Далее мы можем использовать готовое программное обеспечение или составить свою собственную программу. Рассмотрим второй вариант.

3. **Построение алгоритма**, с помощью которого на основе модели по исходным данным вычисляется результат.

4. **Составление программы** на языке программирования.

5. **Тестирование и отладка программы**. Цель тестирования — найти как можно больше ошибок в программе. Для этого используют тесты — специально подобранные наборы исходных данных, для которых известен правильный результат. Тестирование не может доказать, что программа правильная, потому что для этого пришлось бы проверить все возможные варианты исходных данных, оно может только выявить ошибки. После того как ошибки найдены, необходимо исправить их — выполнить *отладку* программы.

6. **Выполнение расчетов**. В отличие от тестирования на этом этапе мы выполняем расчеты для тех исходных данных, для которых результаты неизвестны.

7. **Анализ результатов**. После завершения расчетов нужно проверить, не противоречат ли результаты теории и “здравому смыслу”. Например, время движения автомобиля не может быть отрицательным. Если такие противоречия обнаружены, нужно искать ошибку в модели, алгоритме или программе.

Анализ алгоритмов

Задача 1. *Задана следующая операция над двумя произвольными трехзначными десятичными числами.*

1) Записывается результат сложения старших разрядов этих чисел.

2) К нему дописывается результат сложения средних разрядов по такому правилу: если он меньше первой суммы, то полученное число приписывается к первому слева, иначе — справа.

3) Итоговое число получают приписыванием справа к числу, полученному после второго шага, сумму значений младших разрядов исходных чисел.

Какое из перечисленных чисел могло быть построено по этому правилу?

1) 141819
2) 171418
3) 141802
4) 171814

Решение. Заметим, что сумма двух однозначных чисел — это число от 0 до 18 включительно. Все предложенные числа шестизначные, поэтому все суммы, из которых составлены числа, должны быть двузначными.

В варианте 1 встречается число 19, которое не получить суммой двух однозначных чисел, поэтому этот вариант не подходит.

Из условия 2) следует, что первые два двузначных числа должны быть расположены по возрастанию (неубыванию), поэтому вариант 2, где 14 следует за 17, не подходит.

При записи числа 2 ноль впереди не добавляется (в условии про это ничего не сказано), поэтому третий вариант тоже не подходит.

Вариант 4 удовлетворяет всем условиям.

Ответ: 4.

Задача 2. *Автомат получает на вход два двузначных шестнадцатеричных числа. В этих числах все цифры не превосходят цифру 6 (если в числе есть цифра больше 6, автомат отказывается работать). По этим числам строится новое шестнадцатеричное число по следующим правилам.*

1) Вычисляются два шестнадцатеричных числа — сумма старших разрядов полученных чисел и сумма младших разрядов этих чисел.

2) Полученные два шестнадцатеричных числа записываются друг за другом в порядке возрастания (без разделителей).

Пример. Исходные числа: 66, 43. Поразрядные суммы: A, 9. Результат: $9A_{16}$.

Определите, какое из следующих чисел может быть результатом работы автомата.

1) $9F_{16}$
2) 911_{16}
3) 42_{16}
4) $7A_{16}$

Решение. По условию обе цифры числа меньше или равны 6, поэтому при сложении двух таких чисел может получиться сумма от 0 до 12 = C_{16}.

Из условия 1) сразу делаем вывод, что цифры F в записи числа быть не может, вариант 1 не подходит.

Каждая из двух сумм находится на отрезке [0; 12], поэтому записывается одной шестнадцатеричной цифрой, так что результат работы автомата всегда состоит ровно из двух цифр. Поэтому вариант 2, состоящий из трех цифр, не подходит.

По условию цифры записаны в порядке возрастания, поэтому вариант 3 не подходит. Остается вариант 4, для которого все условия соблюдаются.

Ответ: 4.

Задача 3. *Автомат получает на вход четырехзначное число. По этому числу строится новое число по следующим правилам.*

1) Складываются первая и вторая, а также третья и четвертая цифры исходного числа.

2) Полученные два числа записываются друг за другом в порядке убывания (без разделителей).

Пример. Исходное число: 3165. Суммы: 3 + 1 = 4; 6 + 5 = 11. Результат: 114.

Укажите наименьшее число, в результате обработки которого автомат выдаст число 1311.

Решение. Единственный подходящий способ разбить запись 1311 на два числа — это 13 и 11 (числа 131 и 311 не могут образоваться в результате сложения значений двух десятичных цифр). Сумма первой и второй цифр должна быть наименьшей (тогда и число будет меньше!), она равна 11; тогда сумма значений двух последних цифр равна 13.

Для того чтобы все число было минимально, числа, составленные из первых двух и последних двух цифр, должны быть минимальными, соответственно, для сумм 11 и 13. Минимальное двузначное число, у которого сумма значений цифр равна 11, — это 29, с этих двух цифр начинается исходное четырехзначное число. Сумма двух последних цифр равна 13, минимальное двузначное число с такой суммой цифр — 49.

Ответ: 2949.

Бит четности

Задача 4. *В некоторой информационной системе данные кодируются двоичными шестиразрядными словами. При передаче данных возможны искажения, поэтому в конец каждого слова добавляется седьмой (контрольный) разряд — **бит четности** — таким образом, чтобы сумма разрядов нового слова, считая контрольный, была четной. Например, к слову 110011 справа будет добавлен 0, а к слову 101100 — 1.*

После приема слова производится его обработка. При этом проверяется сумма его разрядов, включая контрольный. Если она нечетна, это означает, что при передаче этого слова произошел сбой, и оно автоматически заменяется на зарезервированное слово 0000000. Если она четна, это означает, что сбоя не было или сбоев было больше одного. В этом случае принятое слово не изменяется.

Исходное сообщение

1100101 1001011 0011000

было принято в виде

1100111 1001110 0011000.

Как будет выглядеть принятое сообщение после обработки?

Решение. Сообщение содержит три семибитных блока, по условию в каждом правильно принятом блоке число единиц должно быть четным. В первом принятом блоке 1100111 — нечетное число единиц (5), поэтому он будет заменен на нули 000000. В остальных двух блоках четное число единиц, поэтому принимающая сторона считает их правильными и изменять не будет. Заметим, что второй принятый блок не совпадает с переданным — он отличается в двух битах, но приемник это не обнаруживает.

Ответ: 000000 1001110 0011000.

Задача 5. *На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом.*

1) Строится двоичная запись числа N.

2) К этой записи дописываются справа еще два разряда по следующему правилу:

а) складываются все цифры двоичной записи, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;

б) над этой записью производятся те же действия — справа дописывается остаток от деления суммы цифр двоичной записи числа на 2.

Полученная таким образом запись является двоичной записью искомого числа R. Укажите минимальное число N, после обработки которого с помощью этого алгоритма получается число, большее, чем 137.

Решение. Фактически к двоичной записи числа числу дважды дописывается бит четности. При этом уже после шага а) у нас всегда получится чет-

ное число единиц, поэтому на шаг б) добавляемый бит всегда равен нулю!

Если в конце двоичной записи числа стоит 0, значит, оно четное; поэтому в результате работы алгоритма должно обязательно получиться четное число. Таким образом, мы ищем четное число, большее 137. Это может быть 138, 140 и т.д.

Проверяем число 138: после выполнения шага б) оно увеличилось вдвое (приписали 0), поэтому до выполнения этого шага у нас было число $138 : 2 = 69 = 1000101_2$. В двоичном коде этого числа нечетное число единиц (3), поэтому оно не подходит по условию.

Проверяем следующее число-кандидат: $140 : 2 = 70 = 1000110_2$. В этом числе тоже 3 единицы, оно тоже не подходит.

Следующее четное число, 142, при делении на 2 дает число $71 = 1000111_2$, которое содержит четное число единиц, поэтому оно могло быть получено после шага а) алгоритма. На этом шаге к нему был добавлен бит четности. Убираем последний бит числа 71 (бит четности), получаем $35 = 100011_2$.

Ответ: 35.

Выводы:

- Алгоритмы можно записывать на естественных языках, по шагам, в виде блок-схемы и на языке программирования.
- Этапы решения задач на компьютере: постановка задачи, формализация, построение алгоритма, составление программы, тестирование и отладка программы, выполнение расчетов, анализ результатов.
- Бит четности добавляется к блоку передаваемых данных так, чтобы число единичных битов в расширенном блоке стало четным. Бит четности позволяет обнаружить нечетное количество ошибок.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Зачем нужен бит четности?

2. В каких случаях бит четности позволяет обнаружить ошибку при передаче данных, а в каких — нет?

Задачи

1. Автомат получает на вход трехзначное число. По этому числу строится новое число по следующим правилам.

1) Складываются первая и вторая, а также вторая и третья цифры.

2) Полученные два числа записываются друг за другом в порядке невозрастания без разделителей.

Укажите наименьшее и наибольшее числа, при обработке которых автомат выдает результат

а) 1612
б) 1510
в) 138
г) 1111

2. Автомат получает на вход трехзначное число. По этому числу строится новое число по следующим правилам.

1) Перемножаются первая и вторая, а также вторая и третья цифры.

2) Полученные два числа записываются друг за другом в порядке неубывания без разделителей.

Укажите наименьшее и наибольшее числа, при обработке которых автомат выдает результат

а) 621
б) 1015
в) 915
г) 1228

3. Автомат получает на вход трехзначное число. По этому числу строится новое число по следующим правилам.

1) Складываются первая и вторая, а также вторая и третья цифры исходного числа.

2) Полученные два числа записываются друг за другом в порядке неубывания (без разделителей).

Сколько существует чисел, в результате обработки которых автомат выдаст число

а) 1216
б) 1017
в) 1116
г) 1315

4. Автомат получает на вход трехзначное число. По этому числу строится новое число по следующим правилам.

1) Складываются первая и вторая, а также вторая и третья цифры.

2) Полученные два числа записываются друг за другом в порядке невозрастания (без разделителей).

Сколько существует чисел, в результате обработки которых автомат выдаст число

а) 1715
б) 1613
в) 1511
г) 1312

5. Автомат получает на вход четырехзначное число. По этому числу строится новое число по следующим правилам.

1) Складываются первая и вторая, а также третья и четвертая цифры.

2) Полученные два числа записываются друг за другом в порядке неубывания без разделителей.

Укажите наибольшее число, при обработке которого автомат выдает результат

а) 1214
б) 912
в) 79
г) 1113

6. Автомат получает на вход два трехзначных числа. По этим числам строится новое число по следующим правилам[1]:

1) Вычисляются три числа — сумма старших разрядов заданных трехзначных чисел, сумма средних разрядов этих чисел, сумма младших разрядов.

2) Полученные три числа записываются друг за другом в порядке невозрастания (без разделителей).

[1] Задачи 6–7 предложены Н.Леко.

Какое наименьшее и наибольшее значение может иметь одно из исходных чисел, если

а) второе число равно 694, а в результате работы автомата получено число 11108;

б) второе число равно 486, а в результате работы автомата получено число 13107?

7. Автомат получает на вход два трехзначных числа. По этим числам строится новое число по следующим правилам:

1) Вычисляются три числа — сумма старших разрядов заданных трехзначных чисел, сумма средних разрядов этих чисел, сумма младших разрядов.

2) Полученные три числа записываются друг за другом в порядке неубывания (без разделителей).

Какое наименьшее и наибольшее значение может иметь одно из исходных чисел, если

а) второе число равно 857, а в результате работы автомата получено число 81416;

б) второе число равно 714, а в результате работы автомата получено число 91012?

8. В некоторой информационной системе данные кодируются двоичными шестиразрядными словами с битом четности. После приема все слова, в которых обнаружено нечетное число единичных битов, автоматически заменяются на зарезервированное слово 0000000. Какое сообщение получилось после обработки принятых данных:

а) 1010010 1101001 1010101

б) 1110111 1101111 1010011

в) 1110010 1101011 1010111

г) 1110100 1101101 0010011

9. На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:

1) Строится двоичная запись числа N.

2) К этой записи дописываются справа еще два разряда по следующему правилу:

— складываются все цифры двоичной записи, и остаток от деления суммы на 2 дописывается в конец числа (справа);

— над этой записью производятся те же действия — справа дописывается остаток от деления суммы цифр на 2.

Полученная таким образом запись является двоичной записью искомого числа R. Укажите минимальное число N, после обработки которого с помощью этого алгоритма получается число, большее, чем

а) 103 д) 184

б) 121 е) 190

в) 108 ж) 205

г) 96 з) 137

10. На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:

1) Строится двоичная запись числа N.

2) К этой записи дописываются справа еще два разряда по следующему правилу:

— складываются все цифры двоичной записи, и остаток от деления суммы на 2 дописывается в конец числа (справа);

-— над этой записью производятся те же действия — справа дописывается остаток от деления суммы цифр на 2.

Полученная таким образом запись является двоичной записью искомого числа R. Укажите минимальное число $R > M$, которое может быть получено с помощью этого алгоритма, для M равного

а) 96 д) 156

б) 116 е) 180

в) 130 ж) 185

г) 143 з) 207

Оптимальные линейные программы

Ключевые слова:

- оптимальная программа
- дерево вариантов

Задача 1. *В системе команд исполнителя Удвоитель всего две команды*:

```
1. прибавь 1
2. умножь на 2
```

Программа для исполнителя Удвоитель — это последовательность номеров команд. Составьте самую короткую программу для Удвоителя, которая преобразует число 6 в 28.

Решение. Возможно, в этой простой задаче вы сразу сообразили, какой будет эта самая короткая программа. Но в более сложных случаях это не так просто. Кроме того, нам нужно доказать, что это действительно самая короткая программа. Поэтому рассмотрим общий способ решения таких задач.

В этой задаче нас интересует программа, содержащая меньше всего команд. Самую короткую программу можно назвать *оптимальной по длине*.

> **Оптимальная программа** — это самая лучшая программа по какому-то показателю.

За одну команду мы можем из начального числа 6 получить только число 7 (командой 1) или число 12 (командой 2):

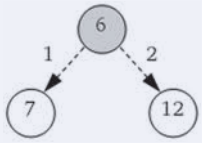


Рис. 1

Проверим все возможные двухшаговые и трехшаговые программы. Мы обнаружим, что ни одна из возможных двухшаговых программ не приводит к нужному нам значению 28, но одна из трехшаговых программ, 122 (она выделена жирными стрелками на *рис.* 2), позволяет решить задачу:

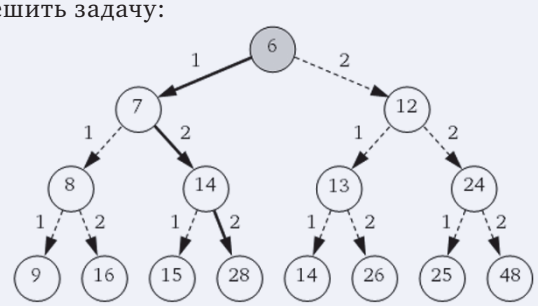


Рис. 2

Поскольку мы перебирали все возможные варианты, можно сделать вывод, что программа 122 из трех команд — оптимальная, то есть самая короткая.

Как видно из *рис.* 2, нет других программ из трех команд, которые приводят к числу 28, поэтому оптимальная программа единственная. Все другие команды, с помощью которых можно получить число 28 из 6, длиннее, чем эта.

Схема, которую мы построили, называется *деревом возможных вариантов*. Действительно, мы рассмотрели *все возможные* программы, состоящие из одного, двух и трех шагов.

Построение полного дерева вариантов при большой длине программы (например, 5 и больше команд) — это очень утомительная работа. Во многих задачах бывает проще двигаться не от начального числа к результату, а наоборот.

Итак, возьмем конечное число 28 и подумаем, какой последней командой мы могли его получить. Так как 28 делится на 2, это могла быть как команда 1 (из 27 получаем 28), так и команда 2 (из 14 также получаем 28). Ни одно из этих чисел не совпадает с начальным (6), поэтому одной командой задачу не решить.

Делаем второй шаг — проверяем, из каких чисел мы могли получить 27 и 14. Число 27 не делится на 2, поэтому оно могло быть получено только командой 1 (из 26), а число 14 можно получить из 13 или из 7:

Рис. 3

До начального числа 6 снова добраться не удалось, поэтому решений из двух команд не существует. Делаем третий шаг:

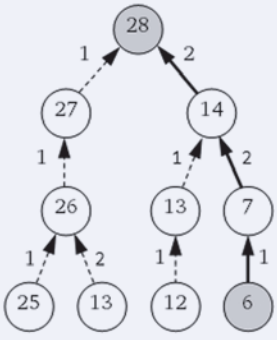


Рис. 4

Для того чтобы получить самую короткую программу, нужно пройти по стрелкам от начального числа к конечному, выписывая встречающиеся номера команд. Для нашей задачи получаем тот же ответ, что и раньше, — 122 (путь выделен жирными стрелками).

Обратите внимание, что дерево при движении “в обратном направлении”, от конечного числа к начальному, получилось меньше. Нам удалось найти решение быстрее, рассматривая меньше вариантов. Это связано с необратимостью одной из команд: любое число можно умножить на 2, но не любое целое число делится на 2 без остатка.

Выводы:

- Оптимальная программа — это самая лучшая программа по какому-то показателю, например, содержащая меньше всего команд.
- Для того чтобы найти оптимальную программу для исполнителя, можно сначала рассмотреть все возможные результаты его работы за один шаг, затем — за два шага и т.д., пока на каком-то шаге не будет получен желаемый результат. Найденная программа для перехода из начального состояния в конечное будет оптимальной по длине (самой короткой).
- Схема, показывающая все возможные результаты работы исполнителя, называется деревом возможных вариантов.
- Если в СКИ исполнителя есть необратимые команды, лучше строить дерево возможных вариантов от конечного состояния к начальному.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Может ли быть так, что задача для Удвоителя решается с помощью нескольких различных программ? Если да, приведите примеры.

2. Как можно доказать, что построенная программа для Удвоителя действительно самая короткая?

3. Какие числа можно (нельзя) получить из натурального числа N с помощью Удвоителя? Из нуля? Из отрицательного числа?

4. Как быстро построить самую короткую программу для получения некоторого числа N из нуля с помощью Удвоителя? Когда эта задача не имеет решений?

5. Исполнитель Калькулятор имеет команды

```
1. прибавь 1
2. раздели на 2
```

Нужно составить самую короткую программу для Калькулятора, с помощью которой из числа a можно получить число b. Как лучше перебирать варианты программ, от начального числа к конечному или наоборот? Почему?

Задачи

1. Удвоителю нужно получить из числа 1 число 4. Сколькими способами это можно сделать? Выпишите все решения в виде программ для Удвоителя.

2. Напишите самую короткую программу для Удвоителя, которая преобразует

а) число 2 в число 42
б) число 3 в число 37
в) число 2 в число 38
г) число 3 в число 65
д) число 5 в число 115
е) число 7 в число 130
ж) число 19 в число 629
з) число 3 в число 63

3. Исполнитель Раздвоитель имеет команды

1. вычти 1
2. раздели на 2

Напишите самую короткую программу для Раздвоителя, которая преобразует

а) число 8 в число 0
б) число 9 в число 0
в) число 16 в число 0
г) число 15 в число 0

4. Исполнитель Калькулятор имеет команды

1. прибавь 1
2. умножь на 3

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 3 в число 34
б) число 4 в число 51
в) число 5 в число 67
г) число 6 в число 70

5. Исполнитель Калькулятор имеет команды

1. прибавь 7
2. раздели на 4

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 20 в число 17
б) число 33 в число 6
в) число 26 в число 31
г) число 17 в число 5

6. Исполнитель Калькулятор имеет команды

1. отними 2
2. раздели на 5

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 177 в число 1
б) число 152 в число 2
в) число 164 в число 4
г) число 190 в число 6

---

7. У исполнителя Калькулятор три команды, которым присвоены номера:

1. прибавь 2
2. прибавь 3
3. умножь на 10

Напишите самую короткую программу для Калькулятора, которая преобразует

число 1 в число 434.

8. Исполнитель Калькулятор имеет команды

1. прибавь 2
2. умножь на x

где *x* — неизвестное положительное число. Известно, что программа 12211 переводит число 1 в число 52. Определите значение *x*.

9. Исполнитель Калькулятор имеет команды

1. прибавь 3
2. умножь на x

где *x* — неизвестное положительное число. Известно, что программа 12112 переводит число 3 в число 36. Определите значение *x*.

10. Исполнитель Калькулятор имеет команды

1. прибавь x
2. умножь на 2

где *x* — неизвестное положительное число. Известно, что программа 12121 переводит число 4 в число 65. Определите значение *x*.

11. Исполнитель Калькулятор имеет команды

1. вычти x
2. умножь на 3

где *x* — неизвестное положительное число. Известно, что программа 12211 переводит число 12 в число 53. Определите значение *x*.

12. Исполнитель Робот ходит по клеткам бесконечной вертикальной клетчатой доски, переходя по одной из команд вверх, вниз, вправо, влево в соседнюю клетку в указанном направлении. Робот выполнил следующую программу:

влево вверх вверх влево
вниз вправо вправо вправо

Укажите наименьшее возможное число команд в программе Робота из той же начальной клетки в ту же конечную.

13. Исполнитель Робот действует на клетчатой доске, между соседними клетками которой могут стоять стены. Робот передвигается по клеткам доски и может выполнять команды 1 (вверх), 2 (вниз), 3 (вправо) и 4 (влево), переходя на соседнюю клетку в направлении, указанном в скобках. Если в этом направлении между клетками стоит стена, то Робот разрушается. Робот успешно выполнил программу

33233241

Какую последовательность из четырех команд должен выполнить Робот, чтобы вернуться в ту клетку, где он был перед началом выполнения программы, и не разрушиться вне зависимости от того, какие стены стоят на поле?

14. У исполнителя Кузнечик, который живет на числовой оси, две команды:

1. вперед N (Кузнечик прыгает вперед на *N* единиц)
2. назад M (Кузнечик прыгает назад на *M* единиц).

Переменные *N* и *M* могут принимать любые целые положительные значения. Кузнечик выполнил программу из 50 команд, в которой команд “**назад 2**” на 12 больше, чем команд “**вперед 3**”. Других команд в программе не было. На какую одну команду можно заменить эту программу, чтобы Кузнечик оказался в той же точке?

15. Исполнитель Кузнечик живет на числовой оси. Начальное положение Кузнечика — точка 10. Система команд Кузнечика:

1. вперед 7
2. назад 4

Какое наименьшее количество раз должна встретиться в программе команда “Назад 4”, чтобы Кузнечик оказался в точке 43?

16. Исполнитель Кузнечик живет на числовой оси. Начальное положение Кузнечика — точка 20. Система команд Кузнечика:

1. вперед 3
2. назад 5

За какое наименьшее количество команд можно перевести Кузнечика в точку “–4”?

17. Исполнитель Чертежник по команде

сместиться на вектор (a, b)

перемещается в точку, в которую можно попасть из данной, пройдя *a* единиц по горизонтали и *b* — по вертикали. Запись:

```
повторить 5 [команда 1 команда 2]
```
означает, что последовательность команд в квадратных скобках повторяется пять раз. Чертежник находился в начале координат и выполнил следующий алгоритм:

```
сместиться на вектор (5, 2)
сместиться на вектор (-3, 3)
повторить 3 [сместиться на вектор (1, 0)]
сместиться на вектор (3, 1)
```

На каком расстоянии от начала координат остановится Чертежник?

18. *У исполнителя, который работает с положительными однобайтовыми двоичными числами, две команды, которым присвоены номера:

```
1. сдвинь влево
2. вычти 1
```

Выполняя первую из них, исполнитель сдвигает число на один двоичный разряд влево и заполняет младший бит нулем, а выполняя вторую, вычитает из него 1. Исполнитель начал вычисления с числа 91 и выполнил цепочку команд 112112. Какое число он получил в результате?

19. *У исполнителя, который работает с положительными однобайтовыми двоичными числами, две команды, которым присвоены номера:

```
1. сдвинь вправо
2. прибавь 4
```

Выполняя первую из них, исполнитель сдвигает число на один двоичный разряд вправо и заполняет старший бит нулем, а выполняя вторую, добавляет к нему 4. Исполнитель начал вычисления с числа 191 и выполнил цепочку команд 112112. Какое число он получил в результате?

Анализ алгоритмов с ветвлениями и циклами

Ключевые слова:

- цикл
- ветвление
- исполнитель Робот
- исполнитель Чертежник
- исполнитель Редактор

Цикл — это многократное выполнение одинаковых действий. Цикл состоит из заголовка и тела цикла — тех команд, которые выполняются несколько раз.

Ветвление — это выбор одного из двух вариантов действий в зависимости от выполнения некоторого условия.

Если хотят, чтобы алгоритм был понятен наибольшему количеству людей, часто записывают его на алгоритмическом языке. Алгоритмический язык — это формальный язык, в нем есть все основные алгоритмические конструкции (ветвления, циклы), которые нужны для однозначной записи алгоритмов. Мы будем использовать алгоритмический язык с русскими служебными словами.

Исполнитель Робот

Система команд исполнителя Робот, “живущего” в прямоугольном лабиринте на клетчатой плоскости: *вверх*, *вниз*, *влево*, *вправо*. При выполнении любой из этих команд Робот перемещается на одну клетку в соответствующем направлении. Четыре логические команды проверяют, есть ли стена у каждой стороны той клетки, где находится Робот:

сверху свободно *снизу свободно*
слева свободно *справа свободно*

Цикл *ПОКА <условие>* выполняется, пока условие истинно, иначе происходит переход на следующую строку.

Задача 1. *Сколько клеток приведенного лабиринта соответствуют требованию, что, выполнив программу*

```
НАЧАЛО
  ПОКА <снизу свободно> вниз
  ПОКА <слева свободно> влево
  ПОКА <сверху свободно> вверх
  ПОКА <справа свободно> вправо
КОНЕЦ
```

Робот уцелеет (не врежется в стену) и остановится в той же клетке, с которой он начал движение?

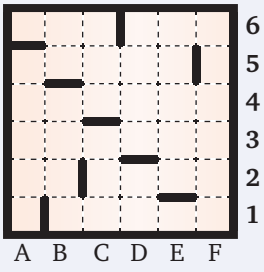


Рис. 5

Решение. Для того чтобы исполнитель вернулся обратно в ту клетку, откуда он начал движения, четыре стенки должны быть расставлены так, чтобы он упирался в них сначала при движении вниз, затем — влево, вверх и, наконец, вправо:

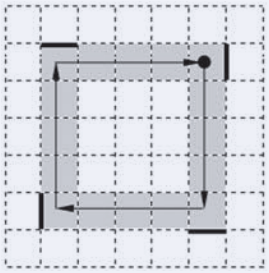


Рис. 6

Точка на *рис.* 6 обозначает клетку, начав движение с которой Робот вернется обратно. Кроме этих четырех стенок, необходимо, чтобы коридор, выделенный на *рис.* 6 фоном, был свободен для прохода. Обратите внимание, что возможны еще “вырожденные” варианты, вроде таких:

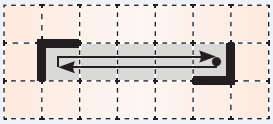


Рис. 7

Итак, нужно рассматривать лишь те клетки-кандидаты, где есть стенка справа. Отметим их на карте:

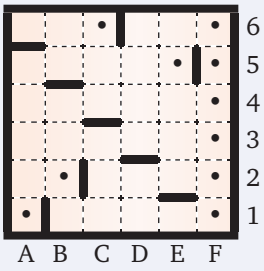


Рис. 8

Количество рассматриваемых клеток можно еще сократить: если Робот начинает движение с любой клетки на вертикали F, он все равно приходит в клетку F4, которая удовлетворяет заданному условию. Таким образом, одну клетку мы нашли, а остальные клетки вертикали F условию не удовлетворяют:

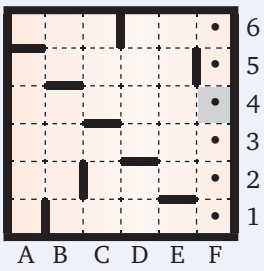


Рис. 9

Проверяем оставшиеся четыре клетки-кандидата: для каждой из них обнаруживаем, что после выполнения алгоритма Робот не приходит в ту клетку, откуда он стартовал. Поэтому условию удовлетворяет только одна клетка — F4.

Ответ: 1.

Исполнитель Чертежник

Исполнитель Чертежник перемещается на координатной плоскости, оставляя след в виде линии. Чертежник может выполнять команду

```
сместиться на (a, b)
```

где a, b — целые числа. Эта команда перемещает Чертежника из точки с координатами (x, y) в точку с координатами $(x + a; y + b)$. Цикл

```
ПОВТОРИ число РАЗ
  последовательность команд
КОНЕЦ
```

означает, что последовательность команд будет выполнена указанное число раз (число должно быть натуральным).

Задача 2. *Чертежнику был дан для исполнения следующий алгоритм (буквами n, a, b обозначены неизвестные числа):*

```
НАЧАЛО
  сместиться на (–2, –11)
  ПОВТОРИ n РАЗ
    сместиться на (a, b)
    сместиться на (27, 12)
  КОНЕЦ
  сместиться на (–22, –7)
КОНЕЦ
```

Укажите все возможные значения n, для которого найдутся такие значения чисел a и b, что после выполнения программы Чертежник возвратится в исходную точку.

Решение. Запишем общее изменение координат Чертежника в результате выполнения этого алгоритма:

$$\Delta x = -2 + n(a + 27) - 22 = n(a + 27) - 24$$

$$\Delta y = -11 + n(b + 12) - 7 = n(b + 12) - 18$$

Поскольку Чертежник должен вернуться в исходную точку, эти величины должны быть равны нулю; следовательно, нужно найти все натуральные значения n, при которых система уравнений

$$\begin{cases} n(a + 27) - 24 = 0 \\ n(b + 12) - 18 = 0 \end{cases} \Leftrightarrow \begin{cases} n(a + 27) = 24 \\ n(b + 12) = 18 \end{cases}$$

разрешима в целых числах относительно a и b. Для этого число n должно быть одновременно делителем чисел 24 и 18. Вот все такие делители: 1, 2, 3, 6.

Исполнитель Редактор

Редактор получает на вход строку символов и преобразовывает ее. Редактор может выполнять две команды, в обеих командах v и w обозначают цепочки символов. Команда

```
нашлось (v)
```

проверяет, встречается ли цепочка v в строке. Если она встречается, то команда возвращает логическое значение “истина”, в противном случае возвращает значение “ложь”. Строка при этом не изменяется. Команда

```
заменить (v, w)
```

заменяет в строке первое слева вхождение цепочки v на цепочку w.

Задача 3. *Дана программа для исполнителя Редактор:*

```
НАЧАЛО
  ПОКА нашлось (222) ИЛИ нашлось (888)
    ЕСЛИ нашлось (222)
      ТО заменить (222, 8)
      ИНАЧЕ заменить (888, 2)
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```

Какая строка получится в результате применения приведенной ниже программы к строке, состоящей из 68 идущих подряд цифр 8?

Решение. Из программы видим, что Редактор что-то делает только тогда, когда в строке есть цепочка 222 или цепочка 888. Если ни одной из этих цепочек нет, программа останавливается.

Если в строке есть 222, то в первую очередь именно эта цепочка меняется (на 8); если в строке нет цепочки 222, но есть 888, то цепочка 888 меняется на 2.

Попробуем выполнить первые шаги алгоритма для цепочки цифр 8. Сначала первые 888 меняются на 2, получается

2 [*65 цифр 8*]

Дальше так же меняем следующие две тройки из цифр 8:

222 [*59 цифр 8*]

Теперь у нас появилась цепочка 222, поэтому в соответствии с алгоритмом она сразу будет заменена на 8, получаем

[*60 цифр 8*]

Таким образом, за первые четыре шага работы цикла мы заменили девять восьмерок на одну или, что то же самое, удалили восемь восьмерок. Очевидно, что следующие четыре шага удалят еще восемь восьмерок и т.д. Мы сможем повторить это восемь раз, после этого останется 68 – 8·8 = 4 восьмерки. На последнем шаге в цепочке 8888 заменяем 888 на 2 и получаем 28.

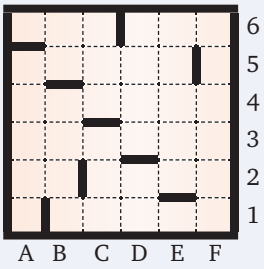
Ответ: 28.

Задачи

1. Сколько клеток приведенного лабиринта соответствуют требованию, что, выполнив предложенную ниже программу, Робот остановится в той же клетке, с которой он начал движение?

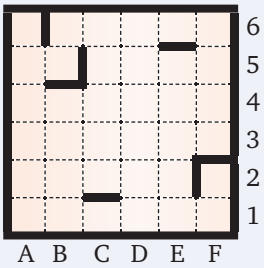
а)

```
НАЧАЛО
  ПОКА <справа свободно> вправо
  ПОКА <сверху свободно> вверх
  ПОКА <слева свободно> влево
  ПОКА <снизу свободно> вниз
КОНЕЦ
```



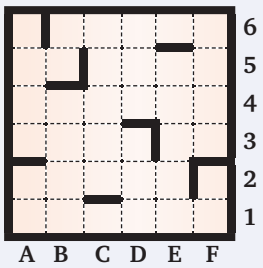
б)

```
НАЧАЛО
  ПОКА <сверху свободно> вправо
  ПОКА <справа свободно> вниз
  ПОКА <снизу свободно> влево
  ПОКА <слева свободно> вверх
КОНЕЦ
```



в)

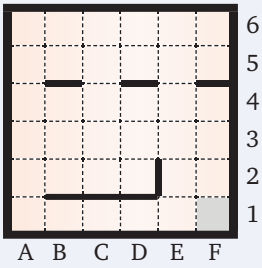
```
НАЧАЛО
  ПОКА <справа свободно> вниз
  ПОКА <снизу свободно> влево
  ПОКА <слева свободно> вверх
  ПОКА <сверху свободно> вправо
КОНЕЦ
```



2. Сколько клеток приведенного лабиринта соответствуют требованию, что, выполнив предложенную ниже программу, Робот остановится в закрашенной клетке (F1)?

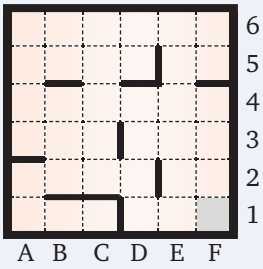
а)

```
НАЧАЛО
  ПОКА <справа свободно ИЛИ
       снизу свободно>
    ПОКА <снизу свободно>
      вниз
    КОНЕЦ
    ПОКА <справа свободно>
      вправо
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```



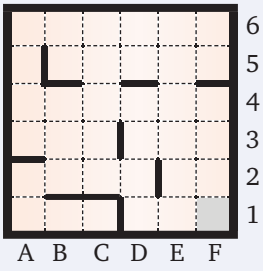
б)

```
НАЧАЛО
  ПОКА <справа свободно ИЛИ
       снизу свободно>
    ПОКА <снизу свободно>
      вниз
    КОНЕЦ
    ПОКА <справа свободно>
      вправо
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```



в)

```
НАЧАЛО
  ПОКА <справа свободно ИЛИ
        снизу свободно>
    вправо
    ПОКА <снизу свободно>
      вниз
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```



3. Чертежнику был дан для исполнения следующий алгоритм:

а)
```
сместиться на (2, -5)
повтори n раз
  сместиться на (25, 12)
  сместиться на (a, b)
конец
сместиться на (–17, –35)
```
б)
```
сместиться на (-7, 5)
повтори n раз
  сместиться на (15, 22)
  сместиться на (a, b)
конец
сместиться на (–17, –35)
```
в)
```
сместиться на (52, -7)
повтори n раз
  сместиться на (15, 22)
  сместиться на (a, b)
конец
сместиться на (–17, –35)
```
г)
```
сместиться на (38, -12)
повтори n раз
  сместиться на (17, 12)
  сместиться на (a, b)
конец
сместиться на (–16, –21)
```
д)
```
сместиться на (38, -12)
повтори n раз
  сместиться на (17, 12)
  сместиться на (a, b)
конец
сместиться на (–21, –22)
```
е)
```
сместиться на (32, -35)
повтори n раз
  сместиться на (7, 32)
  сместиться на (a, b)
конец
сместиться на (6, –22)
```
ж)
```
сместиться на (32, -25)
повтори n раз
  сместиться на (7, b)
  сместиться на (a, 32)
конец
сместиться на (-6, 64)
```
з)
```
сместиться на (32, 20)
повтори n раз
  сместиться на (7, b)
  сместиться на (a, 13)
конец
сместиться на (9, 62)
```

Укажите все возможные значения n, для которого найдутся такие значения чисел a и b, что после выполнения программы Чертежник возвратится в исходную точку.

4. Дана программа для исполнителя Редактор:
```
НАЧАЛО
  ПОКА нашлось (222) ИЛИ нашлось (888)
    ЕСЛИ нашлось (222)
      ТО заменить (222, 8)
      ИНАЧЕ заменить (888, 2)
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```
Какая строка получится в результате применения приведенной ниже программы к строке, состоящей из

а) 62 идущих подряд цифр 8
б) 65 идущих подряд цифр 8
в) 72 идущих подряд цифр 8
г) 93 идущих подряд цифр 8
д) 146 идущих подряд цифр 8
е) 156 идущих подряд цифр 8
ж) 184 идущих подряд цифр 8
з) 193 идущих подряд цифр 8

5. Дана программа для исполнителя Редактор:
```
НАЧАЛО
  ПОКА нашлось (333) ИЛИ нашлось (555)
    ЕСЛИ нашлось (555)
      ТО заменить (555, 3)
      ИНАЧЕ заменить (333, 5)
    КОНЕЦ
  КОНЕЦ
КОНЕЦ
```
Какая строка получится в результате применения приведенной ниже программы к строке, состоящей из

а) 62 идущих подряд цифр 5
б) 65 идущих подряд цифр 5
в) 72 идущих подряд цифр 5
г) 93 идущих подряд цифр 5
д) 146 идущих подряд цифр 5
е) 156 идущих подряд цифр 5
ж) 184 идущих подряд цифр 5
з) 193 идущих подряд цифр 5

Введение в язык Python

Ключевые слова:
- скрипт
- комментарий
- переменная
- тип переменной
- оператор присваивания
- арифметическое выражение

Переменная — это величина, значение которой можно изменять во время работы алгоритма.

Переменная имеет имя, тип и значение. Для изменения значения переменной используется оператор присваивания.

Язык программирования Python

В основной школе вы изучали основы программирования, используя один из учебных языков. Скорее всего это был школьный алгоритмический язык или язык Паскаль. В этом году мы кратко

познакомимся еще с одним языком, который называется Python (или по-русски — Питон). Язык Python — это профессиональный язык программирования, который используется в таких компаниях, как Яндекс и Google. На Python разрабатываются сайты и веб-сервисы, он используется для составления скриптов (от англ. *script* = *сценарий*) — небольших программ, расширяющих возможности других программ, таких, как GIMP, Blender, многие игры.

Простейшая программа

Программы на языке Python чаще всего выполняются *интерпретатором*, который читает очередную команду и сразу ее выполняет, не переводя всю программу в машинный код конкретного процессора. Можно работать в двух режимах:

- через командную строку (в *интерактивном* режиме), когда каждая введенная команда сразу выполняется;
- в программном режиме, когда программа сначала записывается в файл (обычно имеющий расширение **.py**) и при запуске выполняется целиком; такая программа на Python называется *скриптом*.

Мы будем говорить главным образом о программном режиме.

Пустая программа — это программа, которая ничего не делает, но удовлетворяет требованиям выбранного языка программирования. Пустая программа на Python (в отличие от многих других языков программирования) — действительно пустая, она не содержит ни одного *оператора* (команды). Можно добавить в программу *комментарий* — пояснение, которое не обрабатывается транслятором:

```
# Это пустая программа
```

Как вы уже увидели, комментарий начинается знаком "#". Если в программе используются русские буквы, в самом начале обычно записывают специальный комментарий, определяющий кодировку:

```
# -*- coding: utf-8 -*-
```

В данном случае указана кодировка UTF-8.

Команда **print** выводит на экран символы, заключенные в апострофы или в кавычки:

```
print ( "Привет!" )
```

После выполнения этой команды происходит автоматический переход на новую строчку.

Переменные

Напишем программу, которая выполняет сложение двух чисел:

1) запрашивает у пользователя два целых числа;
2) складывает их;
3) выводит результат сложения.

Числа, введенные пользователем, нужно записать в переменные. В языке Python (в отличие от многих других языков) переменные не нужно предварительно объявлять. Они создаются в памяти при первом использовании, точнее, при первом присваивании им значения. Например, при выполнении оператора присваивания

```
a = 4
```

в памяти создается новая переменная (объект типа "целое число") и она связывается с именем **a**. По этому имени теперь можно будет обращаться к переменной: считывать и изменять ее значение.

В именах переменных можно использовать латинские буквы[2] (строчные и заглавные буквы различаются), цифры (но имя не может начинаться с цифры, иначе транслятору будет сложно различить, где начинается имя, а где — число) и знак подчеркивания "_".

Желательно давать переменным "говорящие" имена, чтобы можно было сразу понять, зачем нужна та или иная переменная.

Тип переменной в Python определяется автоматически. Программа

```
a = 4
print ( type(a) )
```

выдает на экран тип (англ. *type*) переменной **a**:

```
<class 'int'>
```

В данном случае переменная **a** целого типа, на это указывает слово **int** (сокращение от англ. *integer* — целый). Говорят, что переменная **a** относится к классу **int**.

Тип переменной нужен для того, чтобы

- определить область допустимых значений переменной;
- определить допустимые операции с переменной;
- определить, какой объем памяти нужно выделить переменной и в каком формате будут храниться данные.

В языке Python используется *динамическая типизация*, это значит, что тип переменной определяется по значению, которое ей присваивается (а не при объявлении переменной, как в языках Паскаль и Си). Таким образом, в разных частях программы одна и та же переменная может хранить значения разных типов[3].

Вспомним, что нам нужно решить три подзадачи:

- ввести два числа с клавиатуры и записать их в переменные (назовем их **a** и **b**);
- сложить эти два числа и записать результат в третью переменную **c**;
- вывести значение переменной **c** на экран.

Для ввода используется команда[4] **input**, результат работы которой можно записать в переменную, например, так:

```
a = input()
```

При выполнении этой строки система будет ожидать ввода с клавиатуры и, когда пользователь введет число и нажмет клавишу Enter, запишет это значение в переменную **a**. При вызове функции **input** в скобках можно записать сообщение-подсказку:

```
a = input ( "Введите целое число: " )
```

Сложить два значения и записать результат в переменную **c** очень просто:

```
c = a + b
```

Символ "=" — это **оператор присваивания**, с его помощью изменяют значение переменной. Он выполняется следующим образом: вычисляется вы-

[2] Можно использовать и русские буквы, но лучше так не делать.

[3] Обсудите в классе, какие преимущества и недостатки имеет этот подход.

[4] Точнее — функция, как и **print**.

ражение справа от символа “=”, а затем результат записывается в переменную, записанную слева. Поэтому, например, оператор

```
i = i + 1
```

увеличивает значение переменной **i** на 1.

Вывести значение переменной **c** на экран с помощью уже знакомой функции **print**:

```
print ( c )
```

Казалось бы, теперь легко собрать всю программу:

```
a = input()
b = input()
c = a + b
print ( c )
```

— однако, после того как мы запустим ее и введем какие-то числа, допустим, 21 и 33, мы увидим странный ответ 2133. Вспомните, что при нажатии клавиши на клавиатуре в компьютер поступает ее код, то есть код соответствующего символа. И входные данные воспринимаются функцией **input** именно как поток символов. Поэтому в той программе, которая приведена выше, переменные **a** и **b** — это цепочки символов, при сложении этих цепочек (с помощью оператора “+”) программа просто объединяет их — приписывает вторую цепочку в конец первой.

Чтобы исправить эту ошибку, нужно преобразовать символьную строку, которая получена при вводе, в целое число. Это делается с помощью функции **int** (от англ. *integer* — целый):

```
a = int ( input() )
b = int ( input() )
```

Возможен еще один вариант: оба числа вводятся не в разных строках, а в одной строке через пробел. В этом случае ввод выполняется сложнее:

```
a,b = map ( int, input().split() )
```

В этой строке собрано сразу несколько достаточно сложных операций:

input() возвращает строку, которая введена с клавиатуры;

input().split() — эта строка разрезается на части по пробелам; в результате получается набор значений (*список*);

map() применяет какую-то операцию ко всем элементам списка; в нашем случае это функция **int()**, которая превращает каждую строку в целое число.

В результате после работы функции **map** мы получаем новый список, состоящий уже из чисел. Первое введенное число (первый элемент списка) записывается в переменную **a**, а второе — в переменную **b**.

Итак, после того как мы преобразовали введенные значения в формат целых чисел, программа работает правильно — складывает два числа, введенные с клавиатуры. Однако у нее есть два недостатка:

1) перед вводом данных пользователь не знает, что от него требуется (сколько чисел нужно вводить и каких);

2) результат выдается в виде числа, которое означает неизвестно что.

Хотелось бы, чтобы диалог программы с пользователем выглядел так:

```
Введите два целых числа:
2
3
2+3=5
```

Подсказку для ввода вы можете сделать самостоятельно. При выводе результата ситуация несколько усложняется, потому что нужно вывести значения трех переменных и два символа: “+” и “=”. Для этого строится список вывода, элементы в котором разделены запятыми:

```
print ( a, "+", b, "=", c )
```

Мы почти получили нужный результат, но почему-то знаки “+” и “=” отделены лишними пробелами, которые мы “не заказывали”:

```
2 + 3 = 5
```

Действительно, функция **print** вставляет между выводимыми значениями так называемый *разделитель* (или сепаратор, англ. *separator*). По умолчанию разделитель — это пробел, но мы можем его изменить, указав новый разделитель после слова **sep**:

```
print ( a, "+", b, "=", c, sep = "" )
```

Здесь мы установили пустой разделитель (пустую строку). Теперь все работает как надо, лишних пробелов нет.

В принципе можно было бы обойтись и без переменной **c**, потому что элементом списка вывода может быть арифметическое выражение, которое сразу вычисляется, и на экран выводится результат:

```
print ( a, "+", b, "=", a + b, sep = "" )
```

Для того чтобы после выполнения функции **print** программа не переходила на новую строку, можно задать пустую строку в дополнительном аргументе **end**:

```
print ( "Привет, ", end = "" )
print ( "Вася!" )
```

В Python можно использовать *форматный вывод*: указать общее количество знакомест, отводимое на число. Например, программа

```
a = 123
print ( "{:5d}".format(a) )
```

выведет значение целой переменной **a**, заняв ровно пять знакомест:

```
◦◦123
```

Поскольку само число занимает только три знакоместа, перед ним выводятся два пробела, которые обозначены как “◦”. Фигурные скобки в строке перед словом **format** показывают место, где будет выведено значение, записанное в скобках после этого слова (это аргумент функции **format** — данные, которые ей передаются). Запись “:5d” означает, что нужно вывести целое число в десятичной системе счисления (англ. *decimal* — десятичный) в пяти позициях.

Можно выводить сразу несколько значений, например, так

```
a = 5
print ( "{:5d}{:5d}{:5d}".format(a, a*a, a*a*a) )
```

Значения, которые должна вывести функция **format**, перечислены в скобках через запятую. Результат будет такой:

```
◦◦◦◦5◦◦◦25◦◦125
```

Типы данных

Перечислим основные типы данных в языке Python:

- **int** — целые значения;
- **float** — вещественные значения (могут быть с дробной частью);
- **bool** — логические значения, **True** (истина, "да") или **False** (ложь, "нет");
- **str** — символ или символьная строка, то есть цепочка символов.

Тип переменной определяется в тот момент, когда ей присваивается новое значение. Мы уже видели, что для определения типа переменной можно использовать стандартную функцию **type**. Например, программа

```
a = 5
print ( type(a) )
a = 4.5
print ( type(a) )
a = True
print ( type(a) )
a = "Вася"
print ( type(a) )
```

выдаст на экран такой результат:

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Сначала в переменной **a** хранится целое значение 5, и ее тип — целый (**int**). Затем мы записываем в нее вещественное значение 4,5, переменная будет вещественного типа (**float**, от англ. *floating point* — с плавающей точкой). Третье присваивание — логическое значение (**bool**, от англ. *boolean* — булевская величина, в честь Дж. Буля). Последнее значение — символьная строка (**str**, от англ. *string* — строка), которая записывается в апострофах или в кавычках.

Целые переменные в Python могут быть сколь угодно большими (или, наоборот, маленькими, если речь идет об отрицательных числах): транслятор автоматически выделяет область памяти такого размера, который необходим для сохранения результата вычислений. Поэтому в Python легко (в отличие от других языков программирования) точно выполнять вычисления с большим количеством значащих цифр.

Вещественные переменные, как правило, занимают 8 байтов, что обеспечивает точность 15 значащих десятичных цифр. Большинство вещественных чисел хранится в памяти неточно, и в результате операций с ними накапливается вычислительная ошибка. Поэтому для работы с целочисленными данными не стоит использовать вещественные переменные.

Логические переменные относятся к типу **bool** и принимают значения **True** (истина) или **False** (ложь).

Арифметические выражения и операции

Арифметические выражения в Python, как и в других языках программирования, записываются в строчку, без многоэтажных дробей. Они могут содержать числа, имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например,

```
a=(c+5-1)/2*d
```

При определении порядка действий используется *приоритет* (старшинство) операций. Они выполняются в следующем порядке:

- действия в скобках;
- возведение в степень (******), справа налево;
- умножение (*****) и деление (**/**), слева направо;
- сложение и вычитание, слева направо.

Таким образом, умножение и деление имеют одинаковый приоритет, более высокий, чем сложение и вычитание. Поэтому в приведенном примере значение выражения, заключенного в скобки, сначала разделится на 2, а потом — умножится на **d**.

В Python (как и в языке Си) разрешено множественное присваивание. Запись

```
a = b = 0
```

равносильна паре операторов

```
b = 0
a = b
```

Так же, как и в Си, часто используют сокращенную запись арифметических операций:

сокращенная запись	*полная запись*
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`

Если в выражение входят переменные разных типов, в некоторых случаях происходит автоматическое приведение типа к более "широкому". Например, результат умножения целого числа на вещественное — это вещественное число. Переход к более "узкому" типу автоматически не выполняется. Нужно помнить, что результат деления (операции "/") — это вещественное число, даже если делимое и делитель — целые и делятся друг на друга нацело[5].

Часто нужно получить целый результат деления целых чисел и остаток от деления. В этом случае используют, соответственно, операторы "**//**" и "**%**" (они имеют такой же приоритет, как умножение и деление):

```
d = 85
a = d // 10    # = 8
b = d % 10     # = 5
```

Обратим внимание на результат выполнения этих операций для отрицательных чисел. Программа

```
print ( -7 // 2 )
print ( -7 % 2 )
```

выдаст на экран числа "–4" и 1. Дело в том, что с точки зрения теории чисел остаток — это неотрицательное число, поэтому $-7 = (-4) \times 2 + 1$, то есть частное от деления (–7) на 2 равно –4, а остаток равен 1. А в Python (в отличие от многих других языков, например, Паскаля и Си) эти операции выполняются математически правильно.

[5] В некоторых языках, например в Си, это не так: при делении целых чисел получается целое число и остаток отбрасывается.

В Python есть операция возведения в степень, которая обозначается двумя звездочками: “**”. Например, выражение $y = 2x^2 + z^3$ запишется так:

```
y = 2*x**2 + z**3
```

Возведение в степень имеет более высокий приоритет, чем умножение и деление, то есть выполняется раньше.

Выводы:

- Программы (скрипты) на языке Python выполняются интерпретатором.
- Переменные в языке Python не нужно объявлять. Любой переменной в разных частях программы можно присваивать значения разных типов.
- При вводе числовых данных нужно преобразовывать их из символьного формата в числовой.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Зачем нужен тип переменной?
2. Почему желательно выводить на экран подсказку перед вводом данных?
3. Когда, по вашему мнению, можно вычислять результат прямо в операторе вывода, а когда нужно заводить отдельную переменную?
4. Какие типы данных вы знаете?
5. Какие данные записываются в логические переменные?
6. Что такое приоритет операций? Зачем он нужен?
7. В каком порядке выполняются операции, если они имеют одинаковый приоритет?
8. Зачем используются скобки?
9. Чем отличаются операции /, // и % ?
10. Расскажите о проблеме вычисления остатка от деления в различных языках программирования. Обсудите в классе этот вопрос.

Задачи

1. Определите значение переменной b после выполнения следующего фрагмента программы.

а)
```
a = 5
a = a + 6
b = -a
b = a - 2 * b
```
б)
```
a = 5
a = 12 - a * a
b = -a
b = 10 * a - b
```
в)
```
a = 5
b = 5 - 3 * a
b = b / 2 * a
```
г)
```
a = 5
b = 5 + 5 * a
b = b / 2 * a
```
д)
```
a = 7
b = 7 + 3 * a
b = b / 2 * a
```
е)
```
a = -5
b = 5 + 7 * a
b = b / 4 * a
```

2. Определите значение переменной c после выполнения следующего фрагмента программы.

а)
```
a = 8
b = 3 + a * 4
b = (b // 10) + 14
c = (b % 10) + 2
```
б)
```
x = 8 + 2 * 5;
y = (x % 10) + 14
x = (y // 10) + 3
c = x - y
```
в)
```
a = 1819
b = (a // 100) * 10 + 9
c = (10 * b - a) % 100
```
г)
```
a = 6 * 12 + 3
b = (a // 10) + 5
c = (b % 10) + 1
```
д)
```
a = 42
b = 14
a = a // b
b = a * b
c = b // a
```
е)
```
x = 5
y = 7
t = x
x = y % x
c = t + y // t
```

3. Вычислите значение переменной c при $a = 2$ и $b = 3$:

а) `c = a + 1/3`
б) `c = a + 4/2 * 3 + 6`
в) `c = (a + 4)/2 * 3`
г) `c = (a + 4)/(b + 3) * a`

4. Вычислите значение переменной c при $a = 26$ и $b = 6$:

а)
```
c = a % b + b
```
б)
```
c = a // b + a
```
в)
```
b = a // b
c = a // b
```
г)
```
b = a // b + b
c = a % b + a
```
д)
```
b = a % b + 4
c = a % b + 1
```
е)
```
b = a // b
c = a % (b + 1)
```
ж)
```
b = a % b
c = a // (b + 1)
```
з)
```
b = a // b
c = a // (b + 1)
```

5. Выполните предыдущее задание при $a = -22$ и $b = 4$.

6. Напишите программу, которая вводит трехзначное число и разбивает его на цифры. Например, при вводе числа 123 программа должна вывести “1, 2, 3”.

7. Напишите программу, которая находит сумму, произведение и среднее арифметическое трех целых чисел, введенных с клавиатуры. Например, при вводе чисел 4, 5 и 7 мы должны получить ответ

```
4 + 5 + 7 = 16, 4 * 5 * 7 = 140,
(4 + 5 + 7)/3 = 5.333333
```

8. Напишите программу, которая возводит введенное число в степень 10, используя только четыре операции умножения.

Темы сообщений:
а) “Применение языка Python”
б) “Среды программирования на языке Python”
в) “Скрипты на языке Python”
г) “Язык Python на мобильных устройствах”

Интересные сайты:
а) *www.python.org* — сайт разработчиков языка Python.
б) *wingware.com/downloads/wingide-101* — бесплатная среда для разработки программ на языке Python.

Ветвления

Ключевые слова:
- условный оператор
- блок команд
- отступы
- полная форма
- неполная форма
- вложенный условный оператор
- сложное условие

Условный оператор предназначен для выбора из двух вариантов действий. После служебного слова **if** (**если**) записывается некоторое условие; если оно истинно, то выполняется блок операторов, расположенный далее. Вторая (необязательная) часть условного оператора начинается словом **else** (**иначе**), сразу после него записывается блок операторов, которые выполняются, если условие после слова **if** ложно.

Условный оператор

Возможности, описанные в предыдущих параграфах, позволяют писать *линейные* программы, в которых операторы выполняются последовательно друг за другом, и порядок их выполнения не зависит от входных данных.

В большинстве реальных задач порядок действий может несколько изменяться, в зависимости от того, какие данные поступили. Например, программа для системы пожарной сигнализации должна выдавать сигнал тревоги, если данные с датчиков показывают повышение температуры или задымленность.

Для этой цели в языках программирования предусмотрены условные операторы. Например, для того чтобы записать в переменную **M** максимальное из значений переменных **a** и **b**, можно использовать оператор:

```
if a > b:
  M = a
else:
  M = b
```

Если верно (истинно) условие, записанное после ключевого слова **if** (“если”), то выполняются все команды (*блок команд*), которые расположены до слова **else** (“иначе”). Если же условие после **if** неверно (ложно), выполняются команды, стоящие после **else**.

В Python в отличие от других языков важную роль играют сдвиги операторов относительно левой границы (*отступы*). Обратите внимание, что слова **if** и **else** начинаются на одном уровне, а все команды внутренних блоков сдвинуты относительно этого уровня вправо на одно и то же расстояние. Это позволяет не использовать особые ограничители блоков (слова **begin** и **end** в языке Паскаль, фигурные скобки в Си-подобных языках). Для сдвига используют символы табуляции (которые вставляются при нажатии на клавишу Tab) или пробелы.

Если в блоке всего один оператор, иногда бывает удобно записать блок в той же строке, что и ключевое слово **if** (**else**):

```
if a > b: M = a
else:     M = b
```

В приведенных примерах условный оператор записан в полной форме: в обоих случаях (истинно условие или ложно) нужно выполнить некоторые действия. Программа выбора максимального значения может быть написана иначе:

```
M = a
if b > a:
  M = b
```

Здесь использован условный оператор *в неполной форме*: в том случае, когда условие ложно, ничего делать не требуется (нет слова **else** и блока операторов после него).

Поскольку операция выбора максимального из двух значений нужна очень часто, в Python есть встроенная функция **max**[6], которая вызывается так:

```
M = max ( a, b )
```

В этом примере можно использовать особую форму условного оператора в Python:

```
M = a if a > b else b
```

которая работает так же, как и приведенный выше условный оператор в полной форме: записывает в переменную **M** значение **a**, если выполняется условие **a > b**, и значение **b**, если это условие ложно.

Часто при каком-то условии нужно выполнить сразу несколько действий. Например, в задаче сортировки значений переменных **a** и **b** по возрастанию нужно поменять местами значения этих переменных, если **a > b**:

```
if a > b:
   c = a
   a = b
   b = c
```

Все операторы, входящие в блок, сдвинуты на одинаковое расстояние от левого края. Заметим, что в Python в отличие от многих других языков программирования есть множественное присваивание, которое позволяет выполнить эту операцию значительно проще:

```
a, b = b, a
```

Кроме знаков “**<**” и “**>**”, в условиях можно использовать другие знаки отношений: “**<=**” (меньше или равно), “**>=**” (больше или равно), “**==**” (равно, два знака “**=**” без пробела, чтобы отличить от операции присваивания) и “**!=**” (не равно).

[6] Есть также и аналогичная функция **min**, которая выбирает минимальное из двух или нескольких значений.

Внутри условного оператора могут находиться любые операторы, в том числе и другие условные операторы. Например, пусть возраст Андрея записан в переменной **a**, а возраст Бориса — в переменной **b**. Нужно определить, кто из них старше. Одним условным оператором тут не обойтись, потому что есть три возможных результата: старше Андрей, старше Борис и оба одного возраста. Решение задачи можно записать так:

```
if a > b:
  print ( "Андрей старше" )
else:
  if a == b:
    print ( "Одного возраста" )
  else:
    print ( "Борис старше" )
```

Условный оператор, проверяющий равенство, находится внутри блока **иначе** (**else**), поэтому он называется *вложенным* условным оператором. Как видно из этого примера, использование вложенных условных операторов позволяет выбрать один из *нескольких* (а не только из двух) вариантов. Если после **else** сразу следует еще один оператор **if**, можно использовать так называемое "каскадное" ветвление с ключевыми словами **elif** (сокращение от **else-if**): если очередное условие ложно, выполняется проверка следующего условия и т.д.

```
if a>b:
  print ( "Андрей старше" )
elif a==b:
  print ( "Одного возраста" )
else:
  print ( "Борис старше" )
```

Обратите внимание на отступы: слова **if**, **elif** и **else** находятся на одном уровне.

Если в цепочке **if-elif-elif-**... выполняется несколько условий, то срабатывает первое из них. Например, программа

```
if cost<1000:
  print ( "Скидок нет." )
elif cost<2000:
  print ( "Скидка 2%." )
elif cost<5000:
  print ( "Скидка 5%." )
else:
  print ( "Скидка 10%." )
```

при **cost = 1500** выдает "**Скидка 2%.**", хотя условие **cost < 5000** тоже выполняется.

Сложные условия

Предположим, что компания набирает сотрудников, возраст которых от 25 до 40 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: "подходит" он или "не подходит" по этому признаку.

На качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) "И", "ИЛИ" и "НЕ". В языке Python они записываются английскими словами "**and**", "**or**" и "**not**".

Пусть в переменной **v** записан возраст сотрудника. Тогда нужный фрагмент программы будет выглядеть так:

```
if v >= 25 and v <= 40:
  print ( "подходит" )
else:
  print ( "не подходит" )
```

При вычислении сложного логического выражения сначала выполняются отношения (**<**, **<=**, **>**, **>=**, **==**, **!=**), а затем — логические операции в таком порядке: сначала все операции **not**, затем — **and**, и в самом конце — **or** (во всех случаях — слева направо). Для изменения порядка действий используют круглые скобки.

Иногда условия получаются достаточно длинными и их хочется перенести на следующую строку. Сделать это в Python можно двумя способами: использовать обратный слэш (это не рекомендуется):

```
if v>=25 \
       and v<=40:
  ...
```

или взять все условие в скобки (перенос внутри скобок разрешен):

```
if ( v>=25
       and v<=40 ):
  ...
```

В языке Python разрешены двойные неравенства, например,

```
if A < B < C:
  ...
```

означает то же самое, что и

```
if A < B and B < C:
  ...
```

Выводы:

- В записи условного оператора в языке Python после условия и после служебного слова **else** ставится двоеточие. Блоки команд в обеих частях условного оператора записываются со сдвигом.
- Знак операции сравнение "равно" записывается как "==", а знак операции "не равно" — как "!=".
- При записи сложных условий разрешается использовать двойные неравенства.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Чем отличаются разветвляющиеся алгоритмы от линейных?
2. Как вы думаете, почему не все задачи можно решить с помощью линейных алгоритмов?
3. Как вы думаете, хватит ли линейных алгоритмов и ветвлений для разработки любой программы?
4. Почему нельзя выполнить обмен значений двух переменных в два шага:

```
a = b
b = a
```

5. Чем отличаются условные операторы в полной и неполной формах? Как вы думаете, можно ли обойтись только неполной формой?
6. Как организовать выбор из нескольких вариантов?
7. Как определяется порядок вычислений в сложном условии?

Задачи

1. Определите значение переменной *c* после выполнения следующего фрагмента программы.

а)
```
a = 40
b = 10
b = - a/2 * b
if a < b:
  c = b - a
else:
  c = a - 2 * b
```
б)
```
a = 6
b = 15
a = b – a * 2
if a > b:
  c = a + b
else:
  c = b – a
```
в)
```
a = -5
b = 3
a = a - b * 2
if a > b:
  c = b - a
else:
  c = a – b
```
г)
```
a = -5
b = -3
a = a - b * 3
if a > b:
  c = b + a
else:
  c = a – b
```
д)
```
a = 40
b = 10
b = a - 2 * b
if a < 2 * b:
  c = a
else:
  c = b
```
е)
```
a = 120
b = 100
a = a + b/2
if b < a/2:
  c = b + a
else:
  c = b + a/2
```
ж)
```
a = 30
b = 6
a = a/5 * b
if a > b:
  c = a - 4 * b
else:
  c = a + 4 * b
```
з)
```
a = 40
b = 6
a = a * 3/b
if a > b:
  c = a + 5 * b
else:
  c = a - 5 * b
```
и)
```
a = 30
b = 6
a = a * 3 / b
if a < b:
  c = 2 * a - 10 * b
else:
  c = 2 * a + 10 * b
```
к)
```
a = 30
b = 6
a = a * 3/b
if a > b:
  c = 2 * a - 5 * (b + 2)
else:
  c = 2 * a + 5 * (b + 2)
```
л)
```
a = 30
b = 6
a = a * 3/b
if a < b:
  c = 3 * a - 5 * (b + 3)
else:
  c = 3 * a + 5 * (b + 3)
```
м)
```
a = 56
b = 2
k = 5
a = a/8 – b
if a > b:
  c = 10 * a – k * b
else:
  c = 10 * a + k * b
```

2. Покажите, что приведенная программа не всегда верно определяет максимальное из трех чисел, записанных в переменные *a*, *b* и *c*:

```
if a > b: M = a
else:     M = b
if c > b: M = c
else:     M = b
```

Приведите контрпример, то есть значения переменных, при которых в переменной *M* будет получен неверный ответ. Как нужно доработать программу, чтобы она всегда работала правильно?

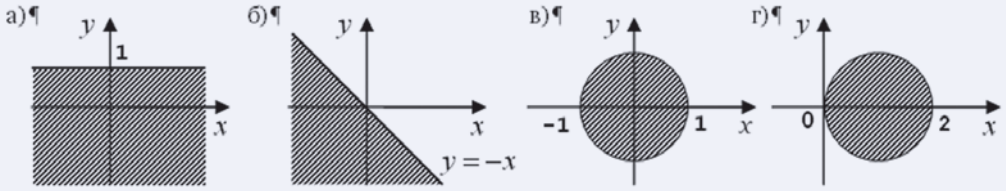
3. Напишите программу, которая выбирает максимальное и минимальное из пяти введенных чисел (не используя встроенные функции **min** и **max**).

4. Напишите программу, которая определяет, верно ли, что введенное число — трехзначное.

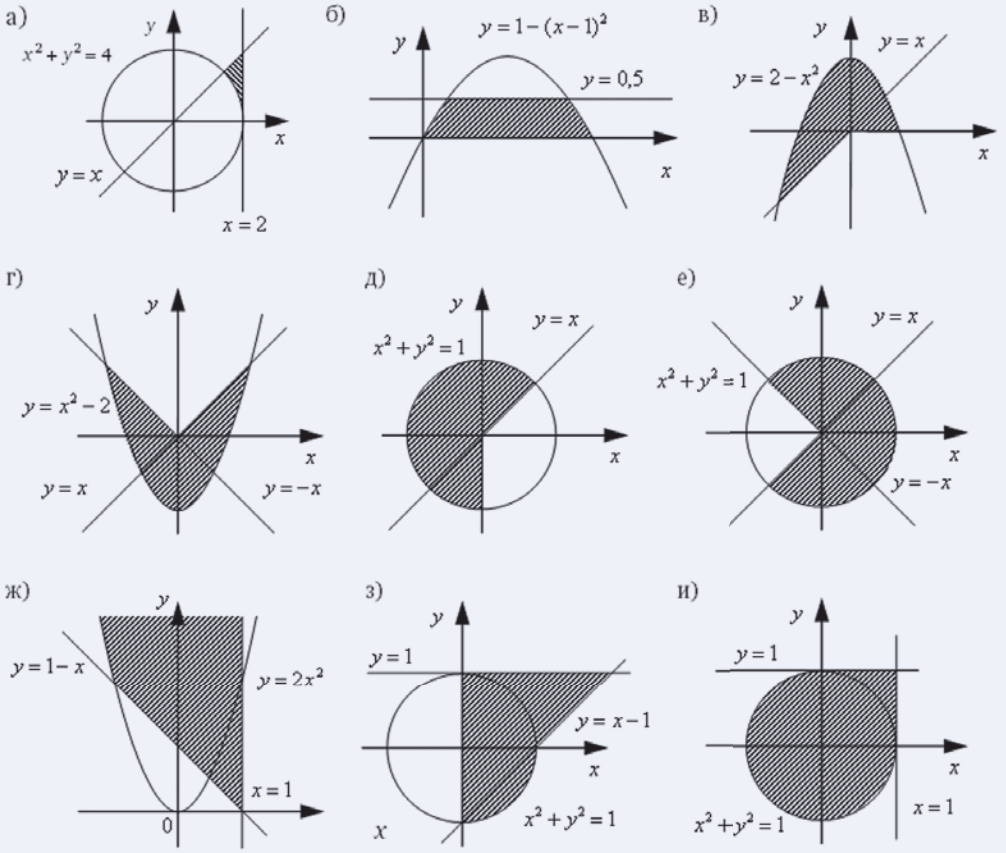
5. Напишите программу, которая вводит номер месяца и выводит название времени года. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.

6. Напишите программу, которая вводит с клавиатуры номер месяца и определяет, сколько дней в этом месяце. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.

7. Напишите программу, которая вводит координаты точки на плоскости и определяет, попала ли эта точка в заштрихованную область.



8. Напишите два варианта программы, которая вводит координаты точки на плоскости и определяет, попала ли эта точка в заштрихованную область. Один вариант программы должен использовать сложные условия, второй — обходиться без них.



9. *Напишите программу, которая вводит с клавиатуры номер месяца и день, и определяет, сколько дней осталось до Нового года. При вводе неверных данных должно быть выведено сообщение об ошибке.

10. *Напишите программу, которая вводит возраст человека (целое число, не превышающее 120) и выводит этот возраст со словом “год”, “года” или “лет”. Например, “21 год”, “22 года”, “25 лет”.

11. *Напишите программу, которая вводит целое число, не превышающее 100, и выводит его прописью, например, 21 → “двадцать один”.

Темы сообщений:

а) “Условные операторы в языке Си”

б) “Условные операторы в языке Javascript”

Циклы с условием

Ключевые слова:

- цикл
- предусловие
- постусловие
- трассировка
- алгоритм Евклида

Цикл — это многократное выполнение одинаковых действий. Любой алгоритм может быть записан с помощью трех алгоритмических конструкций: циклов, условных операторов и последовательного выполнения команд (линейных алгоритмов).

Как организовать цикл?

Подумаем, как можно организовать цикл, который 10 раз выводит на экран слово “привет”. Вы знаете, что программа после запуска выполняется процессором автоматически. И при этом на каждом шаге нужно знать, сколько раз уже выполнен цикл и сколько еще осталось выполнить. Для этого необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла (счетчик шагов). Сначала можно записать в нее ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение ячейки на единицу. Этот алгоритм можно словами записать так (здесь и далее операции, входящие в тело цикла, выделяются отступами):

```
счетчик = 0
пока счетчик != 10
  print ( "Привет!" )
  увеличить счетчик на 1
```

Возможен и другой вариант: сразу записать в счетчик нужное количество шагов, и после каждого шага цикла *уменьшать* счетчик на 1. Тогда цикл должен закончиться при нулевом значении счетчика:

```
счетчик = 10
пока счетчик > 0
  print ( "Привет!" )
  уменьшить счетчик на 1
```

В этих примерах мы использовали *цикл с условием*, который выполняется до тех пор, пока некоторое условие не становится ложно.

Циклы с условием

Рассмотрим следующую задачу: определить количество цифр в десятичной записи целого положительного числа. Будем предполагать, что исходное число записано в переменную n целого типа.

Сначала нужно разработать алгоритм решения задачи. Чтобы подсчитывать что-то в программе, используем переменную, которую называют *счетчиком*. Для подсчета количества цифр нужно как-то отсекать эти цифры по одной, с начала или с конца, каждый раз увеличивая счетчик. Начальное значение счетчика должно быть равно нулю, так как до выполнения алгоритма еще не найдено ни одной цифры.

Для отсечения первой цифры необходимо заранее знать, сколько цифр в десятичной записи числа, то есть нужно заранее решить ту задачу, которую мы решаем. Следовательно, этот метод не подходит.

Отсечь последнюю цифру проще — достаточно разделить число нацело на 10 (поскольку речь идет о десятичной системе). Операции отсечения и увеличения счетчика нужно выполнять столько раз, сколько цифр в числе. Как же “поймать” момент, когда цифры кончатся? Несложно понять, что в этом случае результат очередного деления на 10 будет равен нулю, это и говорит о том, что отброшена последняя оставшаяся цифра. Изменение переменной n и счетчика для начального значения 1234 можно записать в виде таблицы:

Таблица 1

n	счетчик
1234	0
123	1
12	2
1	3
0	4

В словесной форме алгоритм выглядит так:

```
счетчик = 0
пока n > 0
  отсечь последнюю цифру n
  увеличить счетчик на 1
```

А вот соответствующая программа на Python:

```
count = 0
while n > 0:
  n = n // 10
  count += 1
```

Слово **while** переводится как “пока”, то есть цикл выполняется, пока $n > 0$. Переменная-счетчик имеет имя **count**.

Обратите внимание, что проверка условия выполняется в начале очередного шага цикла. Такой цикл называется *циклом с предусловием* (то есть с предварительной проверкой условия) или циклом “пока”. Если в начальный момент значение переменной n будет нулевое или отрицательное, цикл не выполнится ни одного раза.

В данном случае количество шагов цикла “пока” неизвестно, оно равно количеству цифр введенного числа, то есть зависит от исходных данных. Кроме того, этот же цикл может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k = 0
while k < 10:
  print ( "привет" )
  k += 1
```

Если условие в заголовке цикла никогда не нарушится, цикл будет работать бесконечно долго. В этом случае говорят, что “программа зациклилась”. Например, если забыть увеличить переменную k в предыдущем цикле, программа зациклится:

```
k = 0
while k < 10:
  print ( "привет" )
```

Поиск максимальной цифры

Рассмотрим еще один пример использования цикла с условием. Найдем максимальную цифру введенного натурального числа n. В начале параграфа вы уже научились считать цифры числа, используя операции деления, остается немного доработать эту программу.

Максимальную цифру будем хранить в переменной M. Сначала запишем в нее любое значение, меньшее 0, это будет означать, что ни одной цифры еще не найдено. Далее вспомним, что остаток от деления числа на 10 — это последняя цифра в его десятичной записи. Если значение этой цифры больше, чем M, записываем его в M. Когда мы таким образом переберем все цифры числа, в переменной M окажется максимальная цифра введенного числа n. Программа на языке Python запишется в виде цикла, внутри которого стоит условный оператор:

```
n = int(input())
M = -1
while n > 0:
  d = n % 10              # (1)
  if d > M:               # (2)
    M = d                 # (3)
  n = n // 10
print ( M )
```

В строке (1) выделяется последняя цифра числа и записывается в переменную d. В строках (2) и (3) мы изменяем значение M, если только что полученная цифра оказалась больше, чем значение M. Таким образом, в M всегда будет находиться значение максимальной из уже просмотренных цифр исходного числа.

Для проверки работы программы можно использовать **трассировку** (“ручную прокрутку”). Для введенного числа 142 запишем в виде таблицы изменение значений всех переменных после выполнения каждого оператора программы:

Таблица 2

оператор	n	d	M
`n = int(input())`	142		
`M = -1`			-1
`d = n % 10`		2	
`if d > M: M = d`			2
`n = n // 10`	14		
`d = n % 10`		4	
`if d > M: M = d`			4
`n = n // 10`	1		
`d = n % 10`		1	
`if d > M: M = d`			
`n = n // 10`	0		

Обратите внимание, что на последнем шаге цикла значение переменной M не изменилось, потому что цифра 1 не больше, чем значение $M = 4$. Как только значение n стало равно 0, условие работы цикла ($n > 0$) нарушилось и его выполнение закончилось.

Алгоритм Евклида

Древнегреческий математик Евклид, живший в III веке до нашей эры, придумал замечательный алгоритм для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Этот алгоритм и сейчас используется, например, в задачах шифрования. Напомним, что НОД — это наибольшее число, на которое два заданных числа делятся без остатка.

Алгоритм Евклида для натуральных чисел: заменять бóльшее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число и есть их НОД.

Алгоритм Евклида можно записать на языке Python так:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print ( a )
```

Этот вариант алгоритма работает довольно медленно, если одно из чисел значительно меньше другого, например, для пары чисел 2 и 2014. Значительно быстрее выполняется улучшенный (*модифицированный*) алгоритм Евклида.

Модифицированный алгоритм Евклида для натуральных чисел: заменять бóльшее из двух заданных чисел на остаток от деления бóльшего на меньшее, пока этот остаток не станет равен нулю. Тогда второе число и есть их НОД.

Фрагмент программы на языке Python запишется так:

```
while a != 0 and b != 0:
  if a > b:
    a = a % b
  else:
    b = b % a
print ( a + b )
```

Почему в конце выводится на экран сумма `a + b`? Дело в том, что цикл остановится тогда, когда одно из чисел будет равно нулю, тогда второе и есть их НОД. Но так как первое число равно нулю, второе равно их сумме.

Циклы с постусловием

Во многих языках программирования существует *цикл с постусловием*, в котором условие проверяется после завершения очередного шага цикла. Это полезно в том случае, когда нужно обязательно выполнить цикл хотя бы один раз. Например, пользователь должен ввести с клавиатуры положительное число и защитить программу от неверных входных данных.

В языке Python нет цикла с постусловием, но его можно организовать с помощью цикла `while`:

```
print ("Введите положительное число:")
n = int ( input() )
while n <= 0:
print ("Введите положительное число:")
n = int ( input() )
```

Однако такой вариант не очень хорош, потому что нам пришлось написать два раза пару операторов. Но можно поступить иначе:

```
while True:
print ("Введите положительное число:")
n = int ( input() )
if n > 0: break
```

Цикл, который начинается с заголовка `while True`, будет выполняться бесконечно, потому что условие `True` всегда истинно. Выйти из такого цикла можно только с помощью специального оператора `break` (в переводе с англ. — “прервать”, досрочный выход из цикла). В данном случае он сработает тогда, когда станет истинным условие $n > 0$, то есть тогда, когда пользователь введет допустимое значение.

Выводы:

- При записи циклов с условием в языке Python после условия ставится двоеточие. Тело цикла записывается с отступом.
- Алгоритм Евклида для натуральных чисел: заменять бóльшее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число — это их наибольший общий делитель (НОД).
- В Python нет цикла с постусловием, но его можно организовать с помощью цикла с предусловием.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Что означает выражение “цикл с предусловием”?

2. В каком случае цикл с предусловием не выполняется ни разу?

3. В каком случае программа, содержащая цикл с условием, может зациклиться?

4. В каком случае можно заменить цикл с условием на цикл с переменной?

5. Как будет работать приведенная программа, которая считает количество цифр введенного числа, при вводе отрицательного числа? Если вы считаете, что она работает неправильно, укажите, как ее нужно доработать.

Задачи

1. Найдите ошибку в программе:

```
k = 0
while k < 10:
  print ( "привет" )
```

Как ее можно исправить?

2. Определите, сколько раз выполнится этот цикл:

а)
```
n = 0
while n < 5:
  print ( n )
  n = n + 1
```
б)
```
n = 3
while n <= 25:
  print ( n )
  n += 4
```
в)
```
n = 8
while n < 155:
  print ( n )
  n = n + 12
```
г)
```
n = 0
while n <= 15:
  print ( n )
  n -= 2
```
д)
```
n = 12
while n > 5:
  print ( n )
  n = n - 2
```
е)
```
n = 120
while n >= 10:
  print ( n )
  n -= 5
```
ж)
```
n = 103
while n > 15:
  print ( n )
  n = n - 8
```
з)
```
n = 98
while n >= 5:
  print ( n )
  n += 3
```

3. Определите значение переменной *s* после выполнения следующего фрагмента программы.

а)
```
n = 1
s = 0
while n <= 101:
  s = s + 7
  n = n + 1
```
б)
```
n = 4
s = 0
while n <= 8:
  s += 15
  n += 1
```
в)
```
n = 0
s = 512
while s >= 0:
  s = s - 20
  n = n + 1
```
г)
```
n = 14
s = 0
while n <= 18:
  s += 25
  n += 1
```
д)
```
n = 0
s = 1
while s <= 1000:
  s = s * 3
  n = n + 3
```
е)
```
n = 1
s = 0
while s <= 365:
  s += 36
  n *= 2
```
ж)
```
n = 1
s = 0
while n <= 650:
  s = s + 20
  n = n * 5
```
з)
```
n = 1
s = 0
while n <= 300:
  s += 30
  n *= 5
```
и)
```
s = 0
k = 0
while k < 12:
  s = s + 2 * k
  k = k + 3
```
к)
```
s = 0
k = 0
while s < 80:
  s += 2 * k
  k += 4
```

4. Определите наибольшее и наименьшее значения числа `d`, при которых после выполнения программы будет напечатано значение, равное `R`:

а)
```
R = 67
d = int(input())
n = 2
s = 0
while s <= 365:
  s = s + d
  n = n + 5
print(n)
```

б)
```
R = 89
d = int(input())
n = 5
s = 83
while s <= 1200:
  s += d
  n += 6
print(n)
```
в)
```
R = 63
d = int(input())
n = 3
s = 57
while s <= 1200:
  s = s + d
  n = n + 4
print(n)
```
г)
```
R = 150
d = int(input())
n = 3
s = 38
while s <= 1200:
  s += d
  n += 7
print(n)
```
д)
```
R = 121
d = int(input())
n = 1
s = 46
while s <= 2700:
  s = s + d
  n = n + 4
print(n)
```
е)
```
R = 46
d = int(input())
n = 8
s = 78
while s <= 1200:
  s += d
  n += 2
print(n)
```

5. *В результате выполнения фрагмента программы

```
while n != 0:
  print ( 2 * (n % 5) + 3, end='' )
  n = n // 10
```

на экран выведено число 10614. Какое число хранилось в переменной *n* до выполнения этого цикла, если известно, что все цифры в нем нечетные.

6. *В результате выполнения фрагмента программы

```
while n != 0:
  print ( 2 * (n % 10) + 1, end='' )
  n = n // 10
```

на экран выведено число 13717. Укажите все числа, которые могли находиться в переменной *n* до выполнения этого цикла.

7. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких чисел, при вводе которых программа печатает сначала 3, а потом — 12.

а)
```
x = int(input())
a = 0
b = 0
while x > 0:
  a = a + 1
  b = b + (x % 10)
  x = x // 10
print(a, b)
```
б)
```
x = int(input())
a = 0
b = 0
while x > 0:
  a = a + 1
  b = b + (x % 10) // 2
  x = x // 10
print(a, b)
```
в)
```
x = int(input())
a = 0
b = 1
while x > 0:
  a = a + 1
  b = b * (x % 10)
  x = x // 10
print(a, b)
```
г)
```
x = int(input())
a = 0
b = 1
while x > 0:
  a = a + 1
  b = b * (x % 8)
  x = x // 8
print(a, b)
```

8. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких чисел, при вводе которых программа печатает сначала 3, а потом — 20.

а)
```
x = int(input())
a = 0
b = 0
while x > 0:
  a = a + 1
  if x % 2 == 0:
    b = b + (x % 10)
  x = x // 10
print(a, b)
```
б)
```
x = int(input())
a = 0
b = 0
while x > 0:
  a = a + 1
  if x % 2 == 1:
    b = b + 2 * (x % 10)
  x = x // 10
print(a, b)
```

9. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких чисел, при вводе которых программа печатает сначала 3, а потом — 24.

а)
```
x = int(input())
a = 0
b = 1
while x > 0:
  a = a + 1
  if x % 2 == 0:
    b = b * (x % 10)
  x = x // 10
print(a, b)
```

б)
```
x = int(input())
a = 0
b = 1
while x > 0:
  a = a + 1
  if x > 10:
    b = b * (x % 10)
  x = x // 10
print(a, b)
```
10. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких чисел, при вводе которых программа печатает сначала 3, а потом — 5.

а)
```
x = int(input())
a = 0
b = 0
while x > 0:
  a = a + 1
  if b < x % 10:
    b = x % 10
  x = x // 10
print(a, b)
```
б)
```
x = int(input())
a = 0
b = 10
while x > 0:
  a = a + 1
  if b > x % 8:
    b = x % 8
  x = x // 8
print(a, b)
```
11. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких чисел, при вводе которых программа печатает сначала 18, а потом — 2.

а)
```
x = int(input())
a = 0
b = 10
while x > 0:
  c = x % 10
  a = a + c
  if b > c:
    b = c
  x = x // 10
print(a, b)
```
б)
```
x = int(input())
a = 0
b = 10
while x > 0:
  c = x % 10
  a = a + 2 * c
  if b > c:
    b = c - 2
  x = x // 10
print(a, b)
```
12. Ниже записана программа, которая выводит два числа. Укажите наибольшее и наименьшее из таких пятизначных чисел, при вводе которых программа печатает сначала 4, а потом — 3.

а)
```
x = int(input())
a = 0
b = 0
while x > 0:
  c = x % 10
  if c > 3: a = a + 1
  if c < 8: b = b + 1
  x = x // 10
print(a, b)
```
б)
```
x = int(input())
a = 0
b = 0
while x > 0:
  c = x % 10
  if c > 4: a = a + 1
  if c < 7: b = b + 1
  x = x // 10
print(a, b)
```
13. Ниже записана программа. Укажите наименьшее такое число x, большее 100, при вводе которого программа печатает число 26.

а)
```
x = int(input())
a = x
b = 65
if a % 2 == 0:
  b = 52
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print(b)
```
б)
```
x = int(input())
a = x
b = 65
if a % 2 == 0:
  b = 52
while a != 0 and b != 0:
  if a > b:
    a = a % b
  else:
    b = b % a
print(2 * (a + b))
```
14. Напишите программу, которая находит максимальную и минимальную цифры введенного числа.

15. Напишите программу, которая находит среднее арифметическое всех цифр введенного числа.

16. Напишите программу, которая находит количество единиц в двоичной записи введенного числа.

17. Напишите программу, которая находит количество значащих нулей в двоичной записи введенного числа. Например, для числа $83 = 1010011_2$ программа должна вывести 3.

18. Напишите программу, которая находит максимальную и минимальную цифры восьмеричной записи введенного числа.

19. Напишите программу, которая считает количество четных цифр введенного числа.

20. Напишите программу, которая считает сумму цифр введенного числа.

21. *Напишите программу, которая определяет, верно ли, что введенное число содержит две одинаковых цифры, стоящие рядом (как, например, 221).

22. *Напишите программу, которая определяет, верно ли, что введенное число состоит из одинаковых цифр (как, например, 222).

23. *Напишите программу, которая определяет, верно ли, что введенное число содержит по край-

ней мере две одинаковых цифры, возможно, не стоящие рядом (как, например, 212).

24. Напишите программу, которая вычисляет НОД двух целых чисел с помощью алгоритма Евклида, и заполните таблицу в тетради:

a	64168	358853	6365133	17905514	549868978
b	82678	691042	11494962	23108855	298294835
НОД(a,b)					

25. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводятся сумма и произведение введенных чисел (не считая 0).

26. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводятся минимальное и максимальное из введенных чисел (не считая 0).

27. *Напишите программу, которая выводит на экран все цифры числа, начиная с первой.

Темы сообщений:

а) “Циклы с условием в языке Си”

б) “Циклы с условием в языке Javascript”

Циклы по переменной

Ключевые слова:

- цикл
- переменная цикла
- начальное значение
- конечное значение
- шаг
- вложенный цикл
- внешний цикл
- внутренний цикл

Цикл по переменной — это цикл, в котором количество повторения задается переменной цикла. Для переменной цикла указываются начальное и конечное значения, а также шаг изменения.

Что такое цикл по переменной?

Вернемся снова к задаче, которую мы обсуждали в начале параграфа, — вывести на экран 10 раз сообщение “Привет!”. Фактически нам нужно организовать цикл, в котором блок операторов выполнится заданное число раз (в некоторых языках такой цикл есть, например, в школьном алгоритмическом языке он называется “цикл *N* раз”). На языке Python подобный цикл записывается так:

```
for i in range(10):
  print ( "Привет!" )
```

Здесь слово **for** означает “для”, переменная **i** (ее называют переменной цикла) изменяется в диапазоне (**in range**) от 0 до 10, *не включая* 10 (то есть от 0 до 9 включительно). Таким образом, цикл выполняется ровно 10 раз.

В информатике важную роль играют степени числа 2 (2, 4, 8, 16 и т.д.) Чтобы вывести все степени двойки от 2^1 до 2^{10}, мы уже можем написать такую программу с циклом “пока”:

```
k = 1
while k <= 10:
  print ( 2**k )
  k += 1
```

Вы наверняка заметили, что переменная *k* используется трижды (см. выделенные блоки): в операторе присваивания начального значения, в условии цикла и в теле цикла (увеличение на 1). Переменная *k* называется *переменной цикла,* так как ее изменение определяет количество шагов цикла. Цикл по переменной “собирает” все действия с ней в один оператор:

```
for k in range(1,11):
  print ( 2**k )
```

Здесь диапазон (**range**) задается двумя числами — начальным и конечным значением, причем указанное конечное значение *не входит* в диапазон. Такова особенность функции **range** в Python.

Шаг изменения переменной цикла по умолчанию равен 1. Если его нужно изменить, указывают третье (необязательное) число в скобках после слова **range** — нужный шаг. Например, такой цикл выведет только нечетные степени числа 2 (2^1, 2^3 и т.д.):

```
for k in range(1,11,2):
  print ( 2**k )
```

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться. Для этого начальное значение должно быть больше конечного, а шаг — отрицательный. Следующая программа печатает квадраты натуральных чисел от 10 до 1 в порядке убывания:

```
for k in range(10,0,-1):
  print ( k**2 )
```

Обратим внимание, что для конечного значения 0, указанного при вызове функции **range**, цикл уже не будет выполняться.

Вложенные циклы

В более сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция “цикл в цикле”, или “вложенный цикл”.

Предположим, что нужно найти все простые числа в интервале от 2 до 1000. Простейший (но не самый быстрый) алгоритм решения такой задачи выглядит так:

```
for n in range(2,1001):
  if число n простое:
    print ( n )
```

Как же определить, что число простое? Как известно, простое число делится только на 1 и само на себя. Если число *n* не имеет делителей в диапазоне от 2 до *n* – 1, то оно простое, а если хотя бы один делитель в этом интервале найден, то составное.

Чтобы проверить делимость числа *n* на некоторое число *k*, нужно взять остаток от деления *n* на *k*. Если этот остаток равен нулю, то *n* делится на *k*. Таким образом, программу можно записать так (здесь *n*, *k* и *count* — целочисленные переменные, *count* обозначает счетчик делителей):

```
for n in range(2,1001):
  count = 0
  for k in range(2,n):
    if n % k == 0:
      count += 1
  if count == 0:
    print ( n )
```

Попробуем немного ускорить работу программы. Делители числа обязательно идут в парах, причем в любой паре меньший из делителей не превосходит $\sqrt{n}$ (иначе получается, что произведение двух делителей, каждый из которых больше $\sqrt{n}$, будет больше, чем n). Поэтому внутренний цикл можно выполнять только до значения $\sqrt{n}$ вместо $n - 1$. Для того чтобы работать только с целыми числами (и таким образом избежать вычислительных ошибок), лучше заменить условие $k \le \sqrt{n}$ на равносильное ему условие $k^2 \le n$. При этом потребуется перейти к внутреннему циклу с условием:

```
count = 0
k = 2
while k*k <= n:
  if n % k == 0:
    count += 1
  k += 1
```

Чтобы еще ускорить работу цикла, заметим, что когда найден хотя бы один делитель, число уже заведомо составное, и искать другие делители в данной задаче не требуется. Поэтому можно закончить цикл. Для этого при $n \% k == 0$ выполним досрочный выход из цикла с помощью оператора **break**, причем переменная *count* уже не нужна:

```
k = 2
while k*k <= n:
  if n % k == 0: break
  k += 1
if k*k > n:
  print ( n )
```

Если после завершения цикла выполнено условие $k^2 > n$ (нарушено условие в заголовке цикла), то число n простое.

В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
for i in range(1,5):
  for k in range(1,i + 1):
    print ( i, k )
```

На первом шаге (при $i = 1$) переменная k принимает единственное значение 1. Далее, при $i = 2$ переменная k принимает последовательно значения 1 и 2. На следующем шаге при $i = 3$ переменная k проходит значения 1, 2 и 3, и т.д.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Сравните цикл по переменной и цикл с условием. Какие преимущества и недостатки есть у каждого из них?
2. В каком случае цикл по переменной не выполняется ни разу?
3. Верно ли, что любой цикл по переменной можно заменить циклом с условием? Верно ли обратное утверждение?
4. В каком случае можно заменить цикл с условием на цикл по переменной?

Задачи

1. Напишите программу, которая вводит два целых числа и находит их произведение, не используя операцию умножения. Учтите, что числа могут быть отрицательными.
2. Напишите программу, которая вводит натуральное число N и находит сумму всех натуральных чисел от 1 до N.
3. Напишите программу, которая вводит натуральное число N и выводит первые N четных натуральных чисел.
4. Напишите программу, которая вводит натуральные числа a и b и выводит квадраты натуральных чисел на отрезке $[a; b]$. Например, если ввести 4 и 6, программа должна вывести

```
4 * 4 = 16
5 * 5 = 2
6 * 6 = 36
```

5. Напишите программу, которая вводит натуральные числа a и b и выводит сумму квадратов всех натуральных чисел на отрезке $[a; b]$.
6. Найдите все пятизначные числа, которые при делении на 133 дают в остатке 125, а при делении на 134 дают в остатке 111.
7. Напишите программу, которая вводит натуральное число N и выводит на экран все натуральные числа, не превосходящие N и делящиеся на каждую из своих цифр.
8. *Числа Армстронга.* Натуральное число называется числом Армстронга, если сумма цифр числа, возведенных в N-ную степень (где N — количество цифр в числе), равна самому числу. Например, $153 = 1^3 + 5^3 + 3^3$. Найдите все трехзначные и четырехзначные числа Армстронга.
9. *Автоморфные числа.* Натуральное число называется автоморфным, если оно равно последним цифрам своего квадрата. Например, $25^2 = 625$. Напишите программу, которая вводит натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.
10. Напишите программу, которая выводит на экран четные степени числа 2 от 2^{10} до 2^2 в порядке убывания.

Выводы:

- Цикл по переменной используется тогда, когда число повторений цикла заранее известно или может быть вычислено.
- В языке Python изменение переменной цикла задается с помощью функции **range**, аргументы которой — начальное и конечное значения переменной цикла, а также шаг ее изменения. Конечное значение не включается в диапазон перебора.
- Вложенный цикл — это цикл, находящийся внутри другого цикла.
- При работе вложенного цикла значение переменной внутреннего цикла изменяется быстрее, чем значение переменной внешнего цикла.

11. Ряд чисел Фибоначчи задается следующим образом: первые два числа равны 1 ($F_1 = F_2 = 1$), а каждое следующее равно сумме двух предыдущих: $F_n = F_{n-1} + F_{n-2}$. Напишите программу, которая вводит натуральное число N и выводит на экран первые N чисел Фибоначчи.

12. Совершенным называется число, равное сумме всех своих делителей, меньших его самого (например, число 6 = 1 + 2 + 3). Напишите программу, которая вводит натуральное число N и определяет, является ли число N совершенным.

13. Напишите программу, которая вводит натуральные числа a и b и выводит все простые числа в диапазоне от a до b.

14. Напишите программу, которая вводит натуральное число N и находит все совершенные числа в диапазоне от 1 до N.

15. В магазине продается мастика в ящиках по 15, 17, 21 кг. Как купить ровно 185 кг мастики, не вскрывая ящики? Сколькими способами можно это сделать?

Темы сообщений:

а) “Циклы по переменной в языке Си”

б) “Циклы по переменной в языке Javascript”

Функции

Ключевые слова:

- функция
- аргумент функции
- результат функции
- вызов функции
- локальная переменная
- глобальная переменная
- логическая функция

Функция — это вспомогательный алгоритм, который возвращает результат — число, символьную строку или другой объект.

Функция вызывается из основной программы или из другой функции по имени, после которого в скобках через запятую перечисляются ее аргументы — данные, от которых зависит работа функции. Внутри функции эти данные обозначаются именами и называются параметрами функции.

Вместо вызова функции подставляется результат, который она возвращает.

Функции в языке Python

С функциями в языке Python вы уже знакомы. Например, при вводе данных мы использовали функцию **input** — она возвращает символьную строку, введенную с клавиатуры:

```
s = input()
```

Чтобы преобразовать эту строку в число, мы тут вызывали еще одну функцию — **int**, которая получала на вход результат работы функции **input** и возвращала целое число, полученное из введенной строки:

```
n = int( input() )
```

Теперь вы научитесь писать свои собственные функции.

Начнем с простой функции **lastDigit**, которая определяет последнюю цифру переданного ей целого числа. Схема ее работы показана на *рис.* 10.

Рис. 10

Эта функция объявляется так:

```
def lastDigit ( n ):
  d = n % 10
  return d
```

Название функции должно говорить о том, что делает эта функция. Мы выбрали для нее имя **lastDigit**, образованное от английских слов *last* — последний и *digit* — цифра.

Объявление начинается служебным словом **def** (от англ. *define* — определить). В скобках после имени новой функции указан один параметр с именем n. В отличие от многих других языков программирования (например, Паскаля или Си) в Python не нужно указывать тип параметров. Тело функции записывается с отступом, так же как и тело условного оператора или цикла.

После служебного слова **return** (от англ. *return* — вернуть) записывается результат, который возвращает функция. В данном случае результат — это значение переменной d, в которую мы ранее записали последнюю цифру числа.

Теперь эту функцию можно использовать, например, при выводе результатов:

```
n = 4321
print ( 'Число', n, 'оканчивается на',
        lastDigit(n) )
```

или при вычислениях:

```
sum = 0
while n > 0:
  sum += lastDigit(n)
  n = n // 10
```

В этом фрагменте программы определяется сумма цифр числа, записанного в переменную n.

Программы с вызовом функций легче понимать, потому что нам уже не нужно держать в голове алгоритмы, которые используются внутри этих функций, важен только их результат. Это один из способов борьбы с возрастающей сложностью программ.

Внутри функции можно использовать любые конструкции языка, например, циклы и условные операторы. Например, функция **maxDigit** возвращает максимальную цифру переданного ей числа:

```
def maxDigit ( n ):
  M = -1
  while n > 0:
    d = n % 10
    if d > M: M = d
    n = n // 10
  return M
```

Одна функция может вызывать другую. Например, эта функция вычисляет последнюю цифру квадрата числа:

```
def maxDigitX2 ( x ):
  s = maxDigit ( x*x )
  return s
```

Переменные, введенные в теле функции (в последнем примере это переменная s), это *локальные*

переменные, они известны только внутри функции. Другие функции и основная программа о них ничего не знают и использовать не могут.

Переменные, введенные в основной программе, называются *глобальными*, они известны всем функциям. Если глобальную переменную нужно изменять внутри функции, необходимо объявить ее с помощью служебного слова **global** (от англ. *global* — глобальный), иначе будет создана новая локальная переменная с тем же именем:

```
g = 2
def f ( x ):
  global g
  if x < 0:
    g = 1
  return x + g
```

Логические функции

Часто применяют функции, которые возвращают *логическое значение* (**True** или **False**). Иначе говоря, *логическая* функция отвечает на вопрос “да или нет?” и возвращает 1 бит информации.

Построим функцию, которая определяет четность числа. Напомним, что число четное, если остаток от его деления на 2 равен нулю (в этом случае функция должна вернуть логическое значение **True** — истина), и нечетное, если этот остаток — не ноль (тут функция должна вернуть значение **False** — ложь):

```
def even ( n ):
  if n % 2 == 0:
    return True
  else:
    return False
```

Можно записать то же самое несколько проще:

```
def even ( n ):
  return (n % 2 == 0)
```

Здесь **(n%2 == 0)** — это условие, которое может быть истинно или ложно. Это логическое значение и вернет функция.

Вызвать функцию **even** из основной программы можно так:

```
n = int(input())
if even(n):
  print('Число', n, 'четное.')
else:
  print('Число', n, 'нечетное.')
```

Выводы:

- Объявление функции в Python начинается служебным словом **def**. Тело функции записывается с отступом.
- Значение, которое возвращает функция, записывается после служебного слова **return**.
- Переменные, которым присваиваются значения в теле функции, считаются локальными. Если нужно изменить значение глобальной переменной, она описывается с помощью служебного слова **global**.
- Логическая функция возвращает значение **True** (“истина”) или **False** (“ложь”).

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Как по тексту программы определить, какое значение возвращает функция?

2. Какие функции называются логическими? Зачем они нужны?

3. Проверьте с помощью компьютера, какое значение вернет функция, если вы забудете записать в конце оператор **return**.

Задачи

1. Определите, какой результат возвращают эти функции:

а)
```
def f1 ( x ):
  return x * x
```
б)
```
def f2 ( x ):
  return x ** 3 + 5
```
в)
```
def d1 ( x ):
  y = x % 100
  return y // 10
```
г)
```
def d2 ( x ):
  y = x // 10
  return y % 10
```
д)
```
def g1 ( x ):
  if x < 0:
    return -1
  else:
    return 1
```
е)
```
def g2 ( x ):
  if x % 2 == 0:
    return x * x
  else:
    return x ** 3
```
ж)
```
def s ( x ):
  r = 0
  for i in range(1,x + 1):
    r = r + i
  return r
```
з)
```
def p ( x ):
  r = 1
  for i in range(1,x + 1):
    r *= i
  return r
```

2. Для каких значений x функции из предыдущего задания вернут указанные результаты:

а) функция $f_1(x)$ вернет 36?
б) функция $f_2(x)$ вернет 69?
в) функция $d_1(x)$ вернет 5?
г) функция $d_2(x)$ вернет 8?
д) функция $d_2(x)$ вернет 12?
е) функция $g_1(x)$ вернет “–1”?
ж) функция $g_1(x)$ вернет 1?
з) функция $g_1(x)$ вернет 0?
и) функция $g_2(x)$ вернет 144?
к) функция $s(x)$ вернет 6?
л) функция $s(x)$ вернет 15?
м) функция $p(x)$ вернет 6?
н) функция $p(x)$ вернет 1?
о) функция $p(x)$ вернет 120?

3. Заданы две функции. Определите, какой результат возвращает каждая из них:

а)
```
def f1 ( x ):
  return x * x
def g1 ( x ):
  return f1 ( x + 1 )
```
б)
```
def f2 ( x ):
  return x + 5
def g2 ( x ):
  return f2( x + 1 )
```
в)
```
def f3 ( x ):
  return 2 * x
def g3 ( x ):
  return f3 ( x % 10 )
```
г)
```
def f4 ( x ):
  return x % 10
def g4 ( x ):
  return f4( x * x )
```
д)
```
def f5 ( x ):
  if x < 5:
    return 0
  else:
    return 1
def g5 ( x ):
  return f5 ( x % 10 )
```
е)
```
def f6 ( x ):
  return x * x
def g6 ( x ):
  if x > 100:
    return f6((x // 10) % 10)
  else
    return f6( x % 10 )
```

4. Определите результат, который получается при вызове функций из предыдущего задания:

а) $g_1(0)$
б) $g_1(5)$
в) $g_2(0)$
г) $g_2(10)$
д) $g_3(15)$
е) $g_3(250)$
ж) $g_4(7)$
з) $g_4(12)$
и) $g_5(21)$
к) $g_5(75)$
л) $g_6(63)$
м) $g_6(163)$

5. Для каких значений x функции из задания 3 вернут указанные результаты:

а) функция $g_1(x)$ вернет 36?
б) функция $g_1(x)$ вернет 75?
в) функция $g_2(x)$ вернет 21?
г) функция $g_2(x)$ вернет 0?
д) функция $g_3(x)$ вернет 16?
е) функция $g_3(x)$ вернет 15?
ж) функция $g_4(x)$ вернет 6?
з) функция $g_4(x)$ вернет 8?
и) функция $g_5(x)$ вернет 0?
к) функция $g_5(x)$ вернет 1?
л) функция $g_6(x)$ вернет 0?
м) функция $g_6(x)$ вернет 64?

6. Определите, какое число будет выведено на экран:

а)
```
def f ( x ):
  return x * x
k = 100
i = 1
while f(i) < k:
  i = i + 1
print ( i )
```
б)
```
def f ( x ):
  return x * x + 12
k = 93
i = 15
while i > 0 and f(i) >= k:
  i = i - 1
print ( i )
```
в)
```
def f ( x ):
  return x ** 3
k = 216
i = 1
while f(i) <= k:
  i += 1
print ( i )
```
г)
```
def f ( x ):
  return x * x + x + 5
k = 58
i = 12
while i > 0 and f(i) > k:
  i = i - 1
print ( i )
```

7. Определите все возможные значения k, при которых результат работы программы будет такой же, как и при вводе $k = R$:

а)
```
R = 18
def f(x):
  return x * x
i = 0
k = int(input())
while f(i) < k:
  i = i + 1
print(i)
```
б)
```
R = 22
def f(x):
  return x ** 3 + x * x
i = 20
k = int(input())
while i > 0 and f(i) > k:
  i -= 1
print(i)
```
в)
```
R = 28
def f(x):
  return x ** 3 + x * x
i = 0
k = int(input())
while f(i) < k:
  i = i + 1
print(i)
```

г)
```
R = 20
def f(x):
  return x * x + 5 * x
i = 15
k = int(input())
while i > 0 and f(i) > k:
  i -= 1
print(i)
```
д)
```
R = 10
def f(n):
  return n * n * n
def g(n):
  return 2 * n + 3
i = 1
k = int(input())
while f(i) < g(k):
  i = i + 1
print (i)
```
е)
```
R = 16
def f(n):
  return n * n + 5 * n
def g(n):
  return 3 * n + 3
i = 15
k = int(input())
while i > 0 and f(i) > g(k):
  i -= 1
print (i)
```
ж)
```
R = 10
def f(n):
  return n * n
def g(n):
  return 8 * n + 20
i = 1
k = int(input())
while f(i) <= g(k):
  i = i + 1
print (i)
```
з)
```
R = 25
def f(n):
  return n * n * n + 5
def g(n):
  return 13 * n - 7
i = 20
k = int(input())
while i > 0 and f(i) >= g(k):
  i -= 1
print (i)
```

8. *Определите, какое число будет выведено на экран:

а)
```
def f(x):
  return 10 * (5 - x) * (5 - x) + 120
a = -20
b = 20
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) < R:
    M = t
    R = f(t)
print(M)
```
б)
```
def f(x):
    return 19 - 19 * (x - 1) * (x - 1)
a = 10
b = 20
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) < R:
    M = t
    R = f(t)
print(M)
```
в)
```
def f(x):
  return 6 * (x + 10) * (x + 10) + 30
a = -5
b = 20
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) < R:
    M = t
    R = f(t)
print(R)
```
г)
```
def f(x):
  return 281 - 2 * (17 + x) * (17 + x)
a = -7
b = 25
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) > R:
    M = t
    R = f(t)
print(R)
```
д)
```
def f(x):
  return (x * x - 9) * (x * x - 9) + 5
a = -13
b = 13
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) > R:
    M = t
    R = f(t)
print(M + 7)
```
е)
```
def f(x):
  return (x * x - 25) * (x * x - 25) + 15
a = -15
b = 15
M = a
R = f(a)
for t in range(a,b + 1):
  if f(t) <= R:
    M = t
    R = f(t)
print(M + 15)
```

9. Напишите функцию, которая вычисляет максимальное из трех чисел, не используя встроенную функцию **max**.

10. Напишите функцию, которая вычисляет наименьшее общее кратное двух чисел. *Подсказка*: используйте алгоритм Евклида.

11. Напишите функцию, которая вычисляет *N*-е число Фибоначчи.

12. *Напишите функцию, которая "разворачивает" десятичную запись числа наоборот, например, из 123 получается 321, а из 3210 — 123.

13. *Дружественные числа — это два натуральных числа, таких, что сумма всех делителей одного числа (меньших самого этого числа) равна другому числу, и наоборот. Найдите все пары дружественных чисел, каждое из которых меньше 10 000. Используйте функцию, которая вычисляет сумму делителей числа.

14. *Напишите программу, которая вводит натуральное число *N* и находит все числа на отрезке [0; *N*], сумма цифр которых не меняется при умножении на 2, 3, 4, 5, 6, 7, 8 и 9 (например, число 9). Используйте функцию для вычисления суммы цифр числа.

15. *Напишите логическую функцию, которая определяет, верно ли, что число *N* — совершенное, то есть равно сумме своих делителей, меньших него самого.

16. *Напишите логическую функцию, которая возвращает значение `True` ("истина"), если переданное ей число простое (делится только само на себя и на 1).

17. *Простое число называется гиперпростым, если любое число, получающееся из него откидыванием нескольких последних цифр, тоже является простым. Например, число 733 — гиперпростое, так как и оно само, и числа 73 и 7 — простые. Напишите логическую функцию, которая определяет, верно ли, что число *N* — гиперпростое. Используйте функцию, построенную при выполнении предыдущего задания.

Темы сообщений:

а) "Процедуры и функции в языках программирования"

б) "Функции в языке Си"

в) "Функции в языке Javascript"

Рекурсивные функции

Ключевые слова:

- рекурсивная функция
- косвенная рекурсия
- рекуррентная формула
- базовый случай
- алгоритм Евклида
- процедура

Что такое рекурсивная функция?

В предыдущем параграфе мы видели, что одна функция может вызывать другую. В современных языках программирования функция может вызывать и саму себя. Такая функция называется *рекурсивной*.

> **Рекурсивная функция** — это функция, которая вызывает сама себя напрямую или через другие процедуры и функции.

Начнем с примера. Эта функция вычисляет сумму цифр натурального числа:

```
def sumDigits ( n ):
  if n < 10: return n          # (1)
  d = n % 10                   # (2)
  s = d + sumDigits(n // 10)   # (3)
  return s
```

Рассмотрим ее по строкам, используя нумерацию в комментариях.

В самом начале, в строке (1), проверяем особый случай: если натуральное число однозначное, сумма его цифр равна самому числу и можно сразу вернуть результат.

Далее рассуждаем так: *сумма цифр числа равна значению последней цифры плюс сумма цифр числа, полученного из исходного отсечением этой последней цифры*. В строке (2) вычисляем последнюю цифру числа, а в строке (3) записываем только что сформулированный алгоритм на языке Python. При этом функция `sumDigits` вызывает сама себя, но уже для другого аргумента: `n//10` — это число, которое получается после отсечения последней цифры.

Посмотрим, как работает такая функция для числа 123. При первом вызове в локальную переменную *d* записывается значение последней цифры (3). В следующей строке функция вызывает сама себя с аргументом 12. В памяти создается *новый набор локальных переменных s* и *d*, в переменную *d* записывается число 2 и происходит третий вызов функции, уже с аргументом 1.Теперь срабатывает первая строка в функции, возвращается результат 1 и нового вызова не происходит. Этот результат подставляется вместо `sumDigits(1)`, и таким образом второй вызов функции завершает работу с результатом 2 + 1 = 3. Это число подставляется вместо `sumDigits(12)`, после этого первый вызов функции возвращает 3 + 3 = 6.

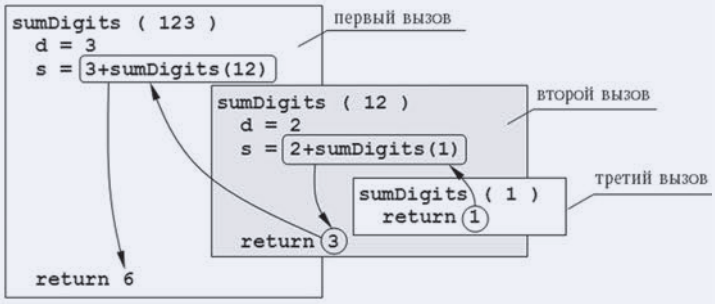


Рис. 11

В этой задаче локальные переменные *d* и *s* введены только для того, чтобы сделать объяснение более понятным. Можно было записать эту функцию в краткой форме:

```
def sumDigits ( n ):
  if n < 10:
    return n
  else:
    return (n % 10) + sumDigits(n // 10)
```

Как использовать рекурсию?

Не всякая задача может быть успешно решена с помощью рекурсии. В чем особенность задачи с вы-

числением суммы цифр числа? Нам удалось свести ее к более простой задаче *того же типа* — к задаче вычисления суммы цифр более короткого числа (*рис.* 12).

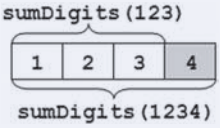


Рис. 12

Если решение задачи имеет такую "вложенную" структуру, рекурсия очень часто бывает полезна.

В более сложных случаях может получиться, что исходная задача включает решение *нескольких* более простых задач того же типа, тогда функция будет вызывать сама себя несколько раз.

При использовании рекурсивных решений нужно помнить про одну особенность — в рекурсивной функции обязательно должен быть *базовый случай* — вариант, при котором нового вызова не происходит. Действительно, если пропустить строчку

```
if n < 10: return n
```

в функции **sumDigits**, при каждом вызове будет происходить новый вызов функции, и эти вызовы (теоретически) никогда не закончатся, точнее, закончатся, когда будет исчерпана доступная память компьютера и программа завершится аварийно. Получится бесконечная рекурсия — серьезная ошибка в программе. Для надежности лучше начинать любую рекурсивную функцию с обработки варианта, когда рекурсия останавливается, как это сделано в нашем примере.

Рекурсия и циклы

Вы могли заметить, что алгоритм вычисления суммы цифр числа — циклический. Если решать эту задачу без использования рекурсии, придется использовать цикл с условием, например, вот так:

```
def sumDigits2 ( n ):
  s = 0
  while n > 0:
    s += n % 10
    n = n // 10
  return s
```

Поэтому можно сделать вывод: рекурсия может заменить цикл.

Однако к такой замене нужно относиться осторожно. С одной стороны, рекурсивное решение задачи часто более понятно, чем решение с циклами. С другой стороны, использование рекурсии часто приводит к увеличению расхода памяти (например, за счет создания новых локальных переменных при каждом вызове). Кроме того, сами дополнительные вызовы функции — это выполнение лишних команд процессора, и поэтому рекурсивное решение часто работает медленнее.

Поэтому если у задачи есть простое решение без рекурсии, программисты всегда предпочитают использовать именно его[7]. А в сложных случаях рекурсия часто помогает упростить решение и сделать его более понятным и, следовательно, более надежным.

[7] Доказано, что рекурсивное решение любой задачи можно заменить на решение без рекурсии, но оно может получиться очень сложным.

Анализ рекурсивных функций

Задача состоит в том, чтобы выяснить, какое значение вернет рекурсивная функция при известных входных данных.

Задача 1. *Определите результат работы программы:*

```
def f ( x ):
  if x < 3:
    return 1
  else:
    return f(x - 1) + 2
print ( f(5) )
```

Решение. Если функция содержит только один рекурсивный вызов, можно использовать прямую подстановку. Как следует из последней строки программы, нас интересует значение $f(5)$. Смотрим в тело функции: условие $5 < 3$ ложно, поэтому срабатывает рекурсивный вызов

```
f(5) = f(4) + 2
```

Далее таким же способом получаем $f(4) = f(3) + 2$, что дает

```
f(5) = f(3) + 2 + 2 = f(3) + 4
```

При следующем вызове $f(3) = f(2) + 2$, так что

```
f(5) = f(3) + 4 = f(2) + 6
```

При $f(2)$ срабатывает условие `x < 3` и функция возвращает 1, так что $f(2) = 1$. Поэтому

```
f(5) = f(2) + 6 = 7.
```

Ответ: 7.

Задача 2. *Определите результат работы программы:*

```
def f(x):
  if x < 3:
    return 1
  else:
    return f(x - 1) + 2 * f(x - 2)
print ( f(5) )
```

Решение. Эту задачу также можно решать подстановкой, как и предыдущую. Но можно использовать и другой способ — составление таблицы значений для первых натуральных чисел. Из программы легко понять, что для вычисления $f(x)$ используется формула:

$$f(x)=\begin{cases}1, & x<3\\ f(x-1)+2f(x-2), & x\ge 3\end{cases}$$

Такие формулы, в которых значение функции в точке x вычисляется через ее значения в других точках, называются *рекуррентными*. Сначала внесем в таблицу базовые случаи (при $x < 3$), для которых значения функции известны и равны 1:

x	1	2	3	4	5
$f(x)$	1	1			

Заполним остальные клетки слева направо, используя вторую часть формулы:

x	1	2	3	4	5
$f(x)$	1	1	3	5	11

Например, $f(5) = f(4) + 2{\cdot}f(3) = 5 + 2{\cdot}3 = 11$ (значения $f(4)$ и $f(3)$ к этому моменту в таблице уже есть).

Ответ: 11.

Задача 3. *Сколько символов "*" будет выведено на экран в результате работы этой программы:*

```
def f ( x ):
  if x > 0: g(x - 1)
def g ( x ):
  print ( '*' )
  if x > 1: f(x - 3)
f ( 11 )
```

Решение. В этой программе две рекурсивные функции, причем ни в одной из них нет оператора `return`. Это значит, что функции не возвращают никакого результата, а просто выполняют какие-то действия, в нашем случае — выводят на экран звездочки. Такие подпрограммы в других языках (например, в языке Паскаль) называются *процедурами*.

Обе функции не вызывают сами себя напрямую. Но можно заметить, что они вызывают друг друга, а поэтому фактически вызывают сами себя через другую функцию. Такой прием называется *косвенной рекурсией*.

В каждой функции есть только один вызов другой функции, поэтому получается такая цепочка вызовов:

$f(11) \to g(10) \to f(7) \to g(6) \to f(3) \to g(2) \to f(-1)$.

Обратим внимание, что в теле функции $f(x)$ вообще ничего не выводится, а при каждом вызове $g(x)$ выводится одна звездочка. Поэтому остается подсчитать, сколько раз управление передается в $g(x)$ — три раза, столько звездочек и будет выведено.

Ответ: 3.

Задача 4. *Сколько символов "*" будет выведено на экран в результате работы этой программы:*

```
def f ( x ):
  if x > 0: g(x - 1) + f(x - 2)
  print ( '*' )
def g ( x ):
  print ( '*' )
  if x > 1: f(x - 3)
f ( 9 )
```

Решение. В отличие от предыдущей задачи здесь функция $f(x)$ вызывает сама себя и еще "парную" функцию $g(x)$, поэтому обойтись линейной цепочкой вызовов не получится.

Будем использовать табличный метод, но результатом работы функции считать количество звездочек, которые она печатает. Функция $f(x)$ печатает одну звездочку при $x \le 0$, а при $x > 0$ — столько же звездочек, сколько $g(x-1)$ и затем $f(x-2)$, а потом еще одну, поэтому рекуррентная формула запишется так:

$$f(x) = \begin{cases} 1, & x \le 0 \\ g(x-1) + f(x-2) + 1, & x > 0 \end{cases}$$

Функция $g(x)$ всегда выводит одну звездочку (первая строка в теле функции), а при $x > 1$ — еще столько звездочек, сколько выводится при вызове $f(x-3)$. Получаем

$$g(x) = \begin{cases} 1, & x \le 1 \\ 1 + f(x-3), & x > 1 \end{cases}$$

Сначала заполняем таблицу для базовых случаев:

x	–1	0	1	2	3	4	5	6	7	8	9
$f(x)$	1	1									
$g(x)$	1	1	1								

Заполним остальные клетки слева направо, используя вторые части формул:

x	–1	0	1	2	3	4	5	6	7	8	9
$f(x)$	1	1	3	3	6	6	11	11	19	19	32
$g(x)$	1	1	1	2	2	4	4	7	7	12	12

Например,

$f(9) = g(8) + f(7) + 1 = 12 + 19 + 1 = 32$.

Ответ: 32.

Задача 5. *Определите результат работы программы:*

```
def f ( n ):
  if n < 2: return 1
  return f(n - 1) + 2 * g(n - 1)
def g ( n ):
  if n < 2: return 1
  return g(n - 1) + f(n - 1)
print( f(5) + g(5) )
```

Решение. В этой задаче две функции, каждая из них вызывает вторую и саму себя. Из последней строки определяем, что результат работы программы — это сумма $f(5)+g(5)$. Применим табличный метод. Изучив функции, построим рекуррентные формулы:

$$f(n) = \begin{cases} 1, & n < 2 \\ f(n-1) + 2g(n-1), & n \ge 2 \end{cases}$$

$$g(n) = \begin{cases} 1, & n < 2 \\ g(n-1) + f(n-1), & n \ge 2 \end{cases}$$

Сначала внесем в таблицу базовый случай (при $n < 2$), для которого значения обеих функций известны и равны 1:

x	1	2	3	4	5
$f(x)$	1				
$g(x)$	1				

Заполним остальные клетки слева направо, используя вторые части формул:

x	1	2	3	4	5
$f(x)$	1	3	7	17	41
$g(x)$	1	2	5	12	29

Ответ: 41 + 29 = 70.

Алгоритм Евклида

Алгоритм Евклида, о котором мы говорили ранее, тоже может быть сформулирован рекурсивно. Алгоритм с вычитанием на языке Python записывается так:

```
def nod ( a, b ):
  if a == b: return a
  else:
    if a > b:
      return nod(a - b, b)
    else:
      return nod(a, b - a)
```

а алгоритм с вычислением остатка — так:

```
def nod ( a, b ):
  if a == 0 or b == 0: return a + b
  else:
    if a > b:
      return nod(a % b, b)
    else:
      return nod(a, b % a)
```

Последний вариант можно еще упростить. Давайте предположим, что $a \geq b$. Тогда в первом условии (которое останавливает рекурсию) можно оставить только вторую часть:

```
if b == 0: return a
```

Результат, полученный после взятия остатка a % b, всегда будет меньше, чем b, поэтому для сохранения свойства $a \geq b$ при следующем рекурсивном вызове достаточно поменять местами аргументы:

```
def nod ( a, b ):
  if b == 0: return a
  else:
    return nod(b, a % b)
```

Заметьте, что такое решение будет работать и тогда, когда при первом вызове $a < b$: в этом случае сначала a и b просто поменяются местами.

Выводы:

- Рекурсивная функция — это функция, которая вызывает сама себя.
- Рекурсия полезна при решении задач, которые можно свести к одной или нескольким более простым задачам того же типа.
- Функция может вызывать сама себя через другую функцию. Такая рекурсия называется косвенной.
- В рекурсивной функции обязательно должен быть базовый случай, когда нового вызова не происходит. Иначе рекурсия никогда не остановится.
- Рекурсия может заменить цикл. Но если у задачи есть простое решение с помощью цикла, как правило, лучше использовать именно его.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Функция А вызывает функцию Б, а функция Б — функцию А и сама себя. Какую из них можно назвать рекурсивной?

2. В каком случае рекурсия никогда не остановится? Как этого избежать?

3. Обсудите в классе достоинства и недостатки рекурсии.

Задачи

1. Алгоритм вычисления функции задан рекуррентной формулой. Определите значение $F(5)$.

а) $F(n) = \begin{cases} 1, & n = 1 \\ F(n-1) \cdot (n+1), & n > 1 \end{cases}$

б) $F(n) = \begin{cases} 1, & n = 1 \\ F(n-1) \cdot (n+2), & n > 1 \end{cases}$

в) $F(n) = \begin{cases} 1, & n = 1 \\ F(n-1) \cdot (2n+1), & n > 1 \end{cases}$

г) $F(n) = \begin{cases} 1, & n = 1 \\ F(n-1) \cdot (2n-1), & n > 1 \end{cases}$

2. Алгоритм вычисления функции задан рекуррентной формулой. Определите значение $F(7)$.

а) $F(n) = \begin{cases} 1, & n < 2 \\ F(n-1) + F(n-2), & n \geq 2 \end{cases}$

б) $F(n) = \begin{cases} 1, & n < 2 \\ 2F(n-1) + F(n-2), & n \geq 2 \end{cases}$

в) $F(n) = \begin{cases} 1, & n < 2 \\ 3F(n-1) - F(n-2), & n \geq 2 \end{cases}$

г) $F(n) = \begin{cases} 1, & n < 2 \\ F(n-1) \cdot F(n-2) + 1, & n \geq 2 \end{cases}$

3. Алгоритмы вычисления функций $F(n)$ и $G(n)$ заданы рекуррентными формулами. Определите значение $F(5) + G(5)$.

а) $F(n) = \begin{cases} 1, & n = 1 \\ F(n-1) + 2G(n-1), & n \geq 2 \end{cases}$

$G(n) = \begin{cases} 1, & n = 1 \\ G(n-1) - 2F(n-1), & n \geq 2 \end{cases}$

б) $F(n) = \begin{cases} 2, & n = 1 \\ F(n-1) - G(n-1), & n \geq 2 \end{cases}$

$G(n) = \begin{cases} 1, & n = 1 \\ G(n-1) + F(n-1), & n \geq 2 \end{cases}$

в) $F(n) = \begin{cases} 1, & n = 1 \\ 2F(n-1) - G(n-1), & n \geq 2 \end{cases}$

$G(n) = \begin{cases} 1, & n = 1 \\ 2G(n-1) + F(n-1), & n \geq 2 \end{cases}$

г) $F(n) = \begin{cases} 1, & n = 1 \\ 2F(n-1) - G(n-1), & n \geq 2 \end{cases}$

$G(n) = \begin{cases} 1, & n = 1 \\ G(n-1) + 2F(n-1), & n \geq 2 \end{cases}$

4. Задана рекурсивная функция. Какое значение выведет эта программа?

а)

```
def f ( x ):
  if x < 2:
    return 1
  else:
    return f(x - 1)
print ( f(4) )
```

б)

```
def f ( x ):
  if x < 2:
    return 1
  else:
    return f(x - 1) + 1
print ( f(5) )
```

в)
```
def f ( x ):
  if x < 3:
    return 1
  else:
    return f(x - 1) + f(x - 2)
print ( f(4) )
```
г)
```
def f ( x ):
  if x < 3:
    return 2
  else:
    return f(x - 1) + 2 * f(x - 2)
print ( f(5) )
```
д)
```
def f ( x ):
  if x < 1:
    return 1
  else:
    return 2 * f(x - 1) + f(x - 3)
print ( f(4) )
```
е)
```
def f ( x ):
  if x < 2:
    return 2
  else:
    return 2 * f(x - 1) + 3 * f(x - 3)
print ( f(4) )
```
5. Определите, сколько символов “*” выведет эта программа:

а)
```
def f ( n ):
  print ( '*' )
  if n > 0:
    f ( n - 2 )
    f ( n // 2 )
f ( 5 )
```
б)
```
def f ( n ):
  print ( '*' )
  if n > 0:
    f ( n - 2 )
    f ( n - 2 )
    f ( n // 2 )
f ( 6 )
```
в)
```
def f ( n ):
  print ( '*' )
  if n > 0:
    print ( '*' )
    f ( n - 2 )
    f ( n // 2 )
f ( 7 )
```
г)
```
def f ( n ):
  print ( '*' )
  if n > 0:
    print ( '*' )
    f ( n - 2 )
    f ( n // 2 )
    f ( n // 2 )
f ( 7 )
```
д)
```
def f ( n ):
  if n > 1:
    f ( n - 2 )
    f ( n - 1 )
    f ( n // 2 )
  print ( '*' )
f ( 7 )
```
е)
```
def f ( n ):
  if n > 2:
    print ( '*' )
    f ( n - 2 )
    f ( n - 1 )
    f ( n // 2 )
  print ( '*' )
f ( 6 )
```
6. *Найдите сумму чисел, которые выведет эта программа:

а)
```
def f ( n ):
  print ( n )
  if n < 7:
    f ( n + 1 )
f ( 2 )
```
б)
```
def f ( n ):
  if n < 6:
    print ( n )
    f ( n + 2 )
f ( 1 )
```
в)
```
def f ( n ):
  print ( n )
  if n < 6:
    f ( n + 1 )
    f ( n + 2 )
f ( 1 )
```
г)
```
def f ( n ):
  if n < 7:
    print ( n )
    f ( n + 1 )
    f ( n * 3 )
f ( 2 )
```
д)
```
def f ( n ):
  print ( n )
  if n < 6:
    print ( n )
    f ( n + 1 )
    f ( n * 2 )
    f ( n * 3 )
f ( 2 )
```
е)
```
def f ( n ):
  print ( n )
  if n < 7:
    print ( n )
    f ( n + 2 )
    f ( n * 2 )
    f ( n * 3 )
f ( 1 )
```
7. *Определите все возможные значения k, при которых результат работы программы будет такой же, как и при вводе $k = R$:

а)
```
R = 45
def f(x):
  if x < 2:
    return 1
  else:
```

```
    return 2 * f(x - 1) + f(x - 2)
i = 0
k = int(input())
while f(i) < k:
  i += 1
print(i)
```
б)
```
R = 120
def f(x):
  if x < 1:
    return 1
  else:
    return f(x - 1) + 3 * f(x - 2)
i = 0
k = int(input())
while f(i) < k:
  i += 1
print(i)
```
в)
```
R = 30
def f(x):
  if x < 3:
    return 1
  else:
    return 2 * f(x - 1) + f(x - 2)
i = 15
k = int(input())
while (i > 0)  and (f(i) > k):
  i -= 1
print(i)
```
г)
```
R = 36
def f(x):
  if x < 2:
    return 1
  else:
    return f(x - 1) + 2 * f(x - 2)
i = 28
k = int(input())
while (i > 0)  and (f(i) > k):
  i -= 1
print(i)
```

8. Напишите рекурсивную функцию, которая вычисляет количество цифр натурального числа.

9. Напишите рекурсивную функцию, которая вычисляет факториал натурального числа. Факториалом числа N (обозначается $N!$) называется произведение всех натуральных чисел от 1 до N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$.

10. Напишите рекурсивную функцию, которая вычисляет число Фибоначчи с номером N.

Темы сообщений:

а) “Задача о Ханойских башнях”
б) “Рекурсия в искусстве и рекламе”
в) “Рекурсивные кривые”
г) “Рекурсивные функции в языке Си”
д) “Рекурсивные функции в языке Javascript”

Массивы

Ключевые слова:

- массив
- список
- перебор элементов
- перестановка элементов
- поиск
- максимальный элемент
- отбор элементов
- сортировка

Массив — это группа переменных одного типа, расположенных в памяти рядом и имеющих общее имя. К элементу массива можно обращаться по индексу (номеру).

Сортировка массива — это перестановка элементов массива в заданном порядке.

Массивы и Python

В языке Python нет такой структуры данных, как “массив”. Вместо этого для хранения группы однотипных[8] объектов используют списки (тип данных `list`).

Список в Python — это набор элементов, каждый из которых имеет свой номер (индекс). Нумерация всегда начинается с нуля (как в Си-подобных языках), второй по счету элемент имеет номер 1 и т.д. В отличие от обычных массивов в большинстве языков программирования список — это динамическая структура данных, его размер можно изменять во время выполнения программы (удалять и добавлять элементы), при этом все операции по управлению памятью берет на себя транслятор.

Список можно создать перечислением элементов через запятую в квадратных скобках, например, так:

```
A=[1, 3, 4, 23, 5]
```

Списки можно “складывать” с помощью знака “+”, например, показанный выше список можно было построить так:

```
A = [1, 3] + [4, 23] + [5]
```

Сложение одинаковых списков заменяется умножением “`*`”. Вот так создается список из 10 элементов, заполненный нулями:

```
A=[0]*10
```

Ввод массива

Далее во всех примерах мы будем работать со списком `A`, состоящим из `N` элементов — целых чисел). В этом списке хранится массив данных, поэтому под выражением “массив” мы будем подразумевать “однотипные данные, хранящиеся в виде списка”. Переменная `i` будет обозначать индекс элемента списка.

Чтобы ввести значения элементов массива с клавиатуры по одному в строке, нужно использовать цикл. Как мы уже говорили, список может изменять свой размер, поэтому можно сначала создать пустой массив

```
A = []
```

а потом добавлять очередной элемент в конец этого массива:

```
N = 20
for i in range(N):
  A.append ( int(input()) )
```

Здесь запись `A.append(...)` означает “добавить в конец массива то, что записано в скобках”, а `int(input())`, как вы помните, — это целое число, введенное с клавиатуры.

[8] И не только однотипных.

Возможен еще один вариант, когда весь массив вводится в одной строке:

```
A = list( map(int, input().split()) )
```

Такая запись означает

1) ввести строку символов с клавиатуры, `input()`;

2) разбить ее на части по пробелам, `input().split()`, в результате получается массив символьных строк, в каждой из которых записано число в символьном виде;

3) применить (`map`) функцию `int` ко всем элементам списка и составить из полученных чисел новый список (объект типа `list`).

Вывод массива

Самый простой способ вывода массива на экран — вывести список как один объект:

```
print ( A )
```

В этом случае весь список берется в квадратные скобки, и элементы разделяются запятыми.

Вывести массив на экран можно и поэлементно

```
for i in range(N):
  print ( A[i], end = "" )
```

После вывода каждого элемента ставится пробел, иначе все значения сольются в одну строку.

Перебор элементов

Перебор элементов массива состоит в том, что мы в цикле просматриваем все элементы списка и, если нужно, выполняем с каждым из них некоторую операцию. Переменная цикла изменяется от **0** до $N - 1$, где N — количество элементов массива, то есть в диапазоне `range(N)`:

```
for i in range(N):
  A[i] += 1
```

В этом примере все элементы массива *A* увеличиваются на 1.

Если массив изменять не нужно, для перебора его элементов удобнее всего использовать такой цикл, в котором все элементы массива *A* по очереди попадают в переменную *x*:

```
for x in A:
  ...
```

Вместо многоточия можно добавлять операторы, работающие с копией элемента в `x`. Обратите внимание, что изменение переменной *x* в теле цикла не приведет к изменению соответствующего элемента массива *A*.

Во многих задачах требуется найти в массиве все элементы, удовлетворяющие заданному условию, и как-то их обработать. Простейшая из таких задач — подсчет нужных элементов. Предположим, что в массиве *A* записаны данные о росте игроков баскетбольной команды. Найдем количество игроков, рост которых больше 180 см, но меньше 190 см. В следующей программе используется переменная-счетчик `count`:

```
count = 0
for x in A:
  if 180 < x and x < 190:
    count += 1
```

Теперь усложним задачу: требуется найти средний рост этих игроков. Для этого нужно дополнительно в отдельной переменной складывать все нужные значения, а после завершения цикла разделить эту сумму на количество. Начальное значение переменной *Sum*, в которой накапливается сумма, тоже должно быть равно нулю.

```
count = 0
Sum = 0
for x in A:
  if 180 < x and x < 190:
    count += 1
    Sum += x
print ( Sum/count )
```

Суммирование элементов массива — это очень распространенная операция, поэтому для суммирования элементов списка в Python существует встроенная функция `sum`:

```
print ( sum(A) )
```

Перестановка элементов массива

Предположим, что нужно поменять местами значения двух элементов массива, $A[i]$ и $A[k]$. Сделать это с помощью двух "очевидных" операторов

```
A[i] = A[k]
A[k] = A[i]
```

не получится — в этом случае оба элемента станут равными значению $A[k]$ до перестановки. Поэтому "старое" значение $A[i]$ нужно сохранить во временной переменной, а затем записать его в $A[k]$:

```
t = A[i]
A[i] = A[k]
A[k] = t
```

В языке Python то же самое можно сделать проще:

```
A[i],A[k] = A[k],A[i]
```

Все нужные операции транслятор выполнит самостоятельно.

Например, переставим соседние элементы массива, $A[0]$ с $A[1]$, $A[2]$ и $A[3]$ и т.д. Предположим, что число элементов массива N — четное. Тогда индекс `i` первого элемента в паре будет изменяться в цикле от 0 до $N-2$ с шагом 2:

```
for i in range(0,N - 1,2):
  A[i],A[i + 1] = A[i + 1],A[i]
```

Обмен значений элементов используется и при сортировке. Например, такой цикл выводит минимальный элемент массива в позицию $A[0]$ (правда, остальные элементы остаются неотсортированными):

```
for i in range(1,N):
  if A[i] < A[0]:
    A[i],A[0] = A[0],A[i]
```

Поиск в массиве

Требуется найти в массиве элемент, равный значению переменной *X*, или сообщить, что его там нет. Алгоритм решения сводится к просмотру всех элементов массива с первого до последнего. Как только найден элемент, равный *X*, нужно выйти из цикла и вывести результат.

```
nX = -1
for i in range (N):
  if A[i] == X:
    nX = i
    break
if nX >= 0:
  print ( "Нашли под номером", nX )
else:
  print ( "Не нашли!" )
```

Для выхода из цикла используется оператор **break**, номер найденного элемента сохраняется в переменной *nX*. Если ее значение осталось равным –1 (не изменилось в ходе выполнения цикла), то в массиве нет элемента, равного *X*.

В языке Python возможен красивый способ решения этой задачи, использующий метод (функцию) **index** для типа данных **list**, которая возвращает номер первого найденного элемента, равного *X*:

```
nX = A.index(X)
```

Тут проблема только в том, что эта строчка вызовет ошибку при выполнении программы, если нужного элемента в массиве нет. Поэтому сначала проверим, есть ли он там (с помощью оператора **in**), а потом используем метод **index**:

```
if X in A:
  nX = A.index(X)
  print ( "A[", nX, "]=", X, sep = "" )
else:
  print ( "Не нашли!" )
```

Запись "**if X in A**" означает "если значение *X* найдено в списке *A*".

Максимальный элемент

Найдем в массиве максимальный элемент. Для его хранения выделим целочисленную переменную *M*. Будем в цикле просматривать все элементы массива один за другим. Если очередной элемент массива больше, чем максимальный из предыдущих (находящийся в переменной *M*), запомним новое значение максимального элемента в *M*.

Остается решить, каково должно быть начальное значение *M*. Во-первых, можно записать туда значение, заведомо меньшее, чем любой из элементов массива. Например, если в массиве записаны натуральные числа, можно записать в *M* ноль или отрицательное число. Если содержимое массива неизвестно, можно сразу записать в *M* значение *A*[*0*], а цикл перебора начать со второго по счету элемента, то есть с *A*[*1*]:

```
M = A[0]
for i in range(1,N):
  if A[i] > M:
    M  = A[i]
print ( M )
```

Для решения этой задачи можно использовать встроенные функции Python: сначала найти максимальный элемент, а потом его индекс с помощью функции **index**:

```
M = max(A)
nMax = A.index(M)
print ( "A[", nMax, "]=", M, sep = "" )
```

Заметим, что в этом случае фактически придется выполнить два прохода по массиву — сначала найти максимум, а потом — его индекс. Однако в практических задачах такой вариант работает достаточно быстро.

Отбор нужных элементов

Требуется отобрать все элементы массива *A*, удовлетворяющие некоторому условию, в новый массив *B*. Поскольку списки в Python могут расширяться во время работы программы, можно использовать такой алгоритм: сначала создаем пустой список, затем перебираем все элементы исходного массива и, если очередной элемент нам нужен, добавляем его в новый список:

```
B = []
for x in A:
  if x % 2 == 0:
    B.append(x)
```

Здесь для добавления элемента в конец списка использован уже знакомый вам метод **append**.

Сортировка массива

В отличие от многих популярных языков программирования в Python есть встроенная функция для сортировки массивов (списков), которая называется **sorted**. Она использует алгоритм *Timsort*[9]. Вот так можно построить новый массив *B*, который совпадает с отсортированным в порядке возрастания массивом *A*:

```
B = sorted ( A )
```

По умолчанию выполняется сортировка по возрастанию (неубыванию). Для того чтобы отсортировать массив по убыванию (невозрастанию), нужно задать дополнительный аргумент **reverse** (от англ. *reverse* — "в обратную сторону"), равный **True**:

```
B = sorted ( A, reverse = True )
```

Иногда требуется особый, нестандартный порядок сортировки, который отличается от сортировок "по возрастанию" и "по убыванию". В этом случае используют еще один именованный аргумент функции **sorted**, который называется **key** (от англ. "ключ"). В этот параметр нужно записать название функции (фактически — ссылку на объект-функцию), которая возвращает число (или символ), используемое при сравнении двух элементов массива.

Предположим, что нам нужно отсортировать числа по возрастанию последней цифры (поставить сначала все числа, оканчивающиеся на 0, затем — все, оканчивающиеся на 1, и т.д.). В этом случае ключ сортировки — это последняя цифра числа, поэтому напишем функцию, которая выделяет эту последнюю цифру:

```
def lastDigit ( x ):
  return x % 10
```

Тогда сортировка массива *A* по возрастанию последней цифры запишется так:

```
B = sorted ( A, key = lastDigit )
```

Функция **sorted** не изменяет исходный массив и возвращает его отсортированную копию. Если нужно отсортировать массив "на месте", лучше использовать метод **sort**, который определен для списков:

```
A.sort ( key = lastDigit, reverse = True)
```

Он имеет те же именованные аргументы, что и функция **sorted**, но изменяет исходный список. В приведенном примере элементы массива *A* будут отсортированы по убыванию последней цифры.

[9] *http://ru.wikipedia.org/wiki/Timsort*.

Выводы:

- Вместо массивов в языке Python используются списки.
- Размер списка может изменяться во время работы программы.
- Для выполнения стандартных операций с массивами (поиска, сортировки и т.п.) в языке Python можно использовать встроенные методы работы со списками.

Самостоятельно составьте интеллект-карту этого параграфа.

Вопросы и задания

1. Зачем нужны массивы?
2. Как вы думаете, почему в языке Python нет массивов, а вместо них используются списки?
3. Как обращаться к отдельному элементу списка?
4. Какие способы ввода списка с клавиатуры вы знаете?
5. Как вывести список на экран? Приведите разные варианты решения этой задачи.
6. Сравните методы решения стандартных задач обработки массивов в языке Python с алгоритмами, которые вы изучали в основной школе.
7. Используя дополнительные источники, выясните, что такое “выход за границы массива”. Чем может быть опасна эта ошибка?

Задачи

1. В массив *A* записаны первые 10 натуральных чисел:

```
A = [1,2,3,4,5,6,7,8,9,10]
```

Определите, какие значения окажутся в массиве *A* после выполнения программы:

а)
```
N = 10
for i in range(N):
  A[9 - i] = A[i]
```
б)
```
N = 10
for i in range(N):
  A[i] = A[9 - i]
```
в)
```
N = 10
for i in range(1,N):
  A[i] = A[i - 1]
```
г)
```
N = 10
for i in range(1,N):
  A[i - 1] = A[i]
```
д)
```
N = 10
t = A[0]
for i in range(N - 1):
  A[i] = A[i + 1]
A[9] = t
```
е)
```
N = 10
t = A[N - 1]
for i in range(N - 1,0,-1):
  A[i] = A[i - 1]
A[0] = t
```
ж)
```
N = 10
for i in range(N):
  t = A[i]
  A[i] = A[9 - i]
  A[9 - i] = t
```
з)
```
for i in range(5):
  t = A[9 - i]
  A[9 - i] = A[i]
  A[i] = t
```
и)
```
for i in range(5):
  t = A[i]
  A[i] = A[i + 5]
  A[i + 5] = t
```
к)
```
for i in range(5):
  t = A[2 * i]
  A[2 * i] = A[2 * i + 1]
  A[2 * i + 1] = t
```

2. В массив *A* записаны 10 натуральных чисел:

```
A = [6,2,8,4,9,7,3,1,10,5]
```

Определите, какие значения окажутся в переменных *c* и *d* после выполнения программы:

а)
```
c = d = 0
N = 10
for i in range(N - 1):
  if A[i] > A[i + 1]:
    c = c + 1
    t = A[i]
    A[i] = A[i + 1]
    A[i + 1] = t
  else:
    d = d + 1
```
б)
```
c = d = 0
N = 10
for i in range(N - 1,0,-1):
  if A[i - 1] > A[i]:
    c = c + 1
    t = A[i]
    A[i] = A[i - 1]
    A[i - 1] = t
  else:
    d = d + 1
```
в)
```
c = 0
N = 10
for i in range(N - 1):
  if A[i] > A[N - 1]:
    c = c + 1
    t = A[i]
    A[i] = A[N - 1]
    A[N - 1] = t
  else:
    d = d + 1
```
г)
```
c = 0
N = 10
for i in range(1,N):
  if A[i] < A[0]:
    c = c + 1
    t = A[i]
    A[i] = A[0]
    A[0] = t
  else:
    d = d + 1
```

д)
```
c = 0
N = 10
for i in range(N - 1):
  if A[i] > A[9]:
    c = c + 1
    t = A[i]
    A[i] = A[i + 1]
    A[i + 1] = t
  else:
    d = d + 1
```
е)
```
c = 0
N = 10
for i in range(1,N):
  if A[i] < A[0]:
    c = c + 1
    t = A[i]
    A[i] = A[i - 1]
    A[i - 1] = t
  else:
    d = d + 1
```
3. В массив *A* записаны 10 натуральных чисел:
```
A = [0,2,4,6,8,10,12,14,16,18]
```
Определите, какое значение окажется в переменной *s* после выполнения программы:

а)
```
s = 0
N = 10
for i in range(N - 1):
  s = s + A[i] - A[i + 1]
```
б)
```
s = 0
N = 10
for i in range(N):
  if A[9 - i] - A[i] < A[i]:
    s = s + A[i]
```
в)
```
s = 0
N = 10
for i in range(N):
  if A[9 - i] - A[i] > 0:
    s = s + A[i]
```
г)
```
s = 0
N = 10
for i in range(N):
  if A[9 - i] - A[i] < 0:
    s = s + 2 * A[i]
```
4. Массив *A* состоит из 10 элементов. Известно, что до обработки в массив была записана возрастающая последовательность чисел, так что $A[0] < A[1] < \ldots < A[9]$. Определите наибольшее возможное значение переменной *s* после выполнения программы:

а)
```
s = 15
N = 10
for i in range(N - 1):
  s += A[i] - A[i + 1]
```
б)
```
s = 32
N = 10
for i in range(N - 1):
  s += A[i] - A[i + 1] + 1
```
5. Массив *A* состоит из 10 элементов. Известно, что до обработки в массив была записана убывающая последовательность чисел, так что $A[0] > A[1] > \ldots > A[9]$. Определите наименьшее возможное значение переменной *s* после выполнения программы:

а)
```
s = 27
N = 10
for i in range(N - 1):
  s += 3 + A[i] - A[i + 1]
```
б)
```
s = 44
N = 10
for i in range(N - 1):
  s += A[i] - A[i + 1] - 5
```
6. Массив *A* состоит из 10 элементов. Известно, что до обработки в массив были записаны трехзначные натуральные числа. Определите наибольшее и наименьшее возможные значения переменной *s* после выполнения программы:

а)
```
s = 3
N = 10
for i in range(N - 1):
  s += A[i] - A[i + 1]
```
б)
```
s = 4
N = 10
for i in range(N - 2):
  s += A[i] - A[i + 2]
```
в)
```
s = 18
N = 10
for i in range(N - 1):
  s += 2 * A[i] - A[i + 1]
```
г)
```
s = 4
N = 10
for i in range(N - 3):
  s += A[i] - 2 * A[i + 3] + 6
```
7. Заполните массив элементами арифметической прогрессии. Ее первый элемент, разность и количество элементов нужно ввести с клавиатуры.

8. Заполните массив степенями числа 2 (от 2^1 до 2^N).

9. Заполните массив первыми числами Фибоначчи.

10. Напишите программу, которая находит максимальный и минимальный из четных положительных элементов массива. Если в массиве нет четных положительных элементов, нужно вывести сообщение об этом.

11. Введите массив с клавиатуры и найдите (за один проход) количество элементов, имеющих максимальное значение.

12. Найдите за один проход по массиву три его различных элемента, которые меньше всех остальных (“три минимума”).

13. *Найдите в массиве все простые числа и скопируйте их в новый массив.

14. *Найдите в массиве все числа Фибоначчи и скопируйте их в новый массив.

Темы сообщений:

а) “Массивы в языке Си”

б) “Массивы в языке Javascript”

Электронные учебники – каждому ученику!

Современный урок уже невозможно представить себе без современных средств обучения. Их функции расширяются год от года. Повседневным становится контроль с моментальной обратной связью. Всё больше входят в наш обиход различные тренинговые программы, позволяющие ученикам быстро развивать навыки работы со сложными объектами. Всё это «вдруг» пришло на уроки ко всем учителям по всем предметам.

Как перейти от удивления от новых возможностей к их осмысленному использованию? Как соединить лучшие наработки традиционной методики с новыми технологическими возможностями?

Как не очароваться новыми возможностями – сохранить баланс между технологией и дидактикой? Дорогие учителя информатики! Вы лучше других педагогов знаете про новые технологии. Ваша экспертиза по вхождению в новую образовательную реальность – использование электронных учебников на уроке и во внеурочной деятельности – бесценна для учителей других предметов. Пожалуйста, делитесь своим опытом.

Дорогие коллеги!

В декабре 2015 года стартовал проект «Оказание услуги по обеспечению доступа обучающихся общеобразовательных организаций Московской области к электронным учебникам и электронным приложениям к учебникам».

Инициатором проекта выступило Министерство образования Московской области. Исполнитель проекта – Издательский дом «Первое сентября».

Целью проекта является предоставление электронных учебников всем учителям и ученикам Московской области. Для этого в течение 2016 г. (до 31 декабря 2016 г. включительно) ученики 5–11-х классов общеобразовательных организаций Московской области могут бесплатно получить электронные учебники на свои устройства (компьютеры, ноутбуки, планшеты), работающие на популярных платформах: Windows, Android (Google), Apple (iOS).

Проект Министерства образования Московской области позволит выяснить желание и готовность учеников учиться с использованием современных образовательных подходов не только в школе, но и дома.

У электронного учебника много преимуществ:

- **наглядность** (этот важный принцип обучения реализуется при демонстрации иллюстраций, проведении лабораторных работ, интерактивном моделировании изучаемых процессов);
- **мультимедийность** (учебный материал подается комплексно, например, термин или слово можно прочитать и послушать, как оно произносится);
- **интерактивность** (например, тренировочные тестовые задания с моментальной обратной связью);
- **гипертекстуальность** (каждый специально обозначенный термин в тексте параграфа можно снабдить подробной дефиницией, открывающейся по клику)
- **удобная навигация**.

С января этого года каждый ученик 5–11-х классов любой из школ Московской области может попросить своего педагога выдать электронные учебники на принесенное из дома устройство.

За два с небольшим месяца с начала проекта доступ к электронной форме учебников по информатике получили 644 ученика и учителя. Надеемся, что их число будет расти.

Использование новых технологий в образовании открывает перед учителем новые методические и дидактические возможности. Но, чтобы воспользоваться этими возможностями, чтобы задействовать их в повседневной работе с учениками, нам всем нужны некоторая доля бесстрашия перед новым и готовность делиться друг с другом своим опытом, который, особенно на первых порах, у каждого свой.

Расскажите о своем опыте работы с электронными учебниками:

- об опыте получения учебника на свои устройства и на школьные устройства,
- об опыте раздачи электронного учебника на устройства учеников,
- об опыте использования учебников на уроке,
- об опыте использования ЭУ учениками,
- об опыте использования ЭУ дома – вне стен школы.

Более подробная информация и страница для обмена опытом на сайте digital.1september.ru/etutorial

Общероссийский проект

Школа цифрового века

Издательский дом «ПЕРВОЕ СЕНТЯБРЯ»

Регистрация школ для участия в проекте в 2016/2017 учебном году открыта!

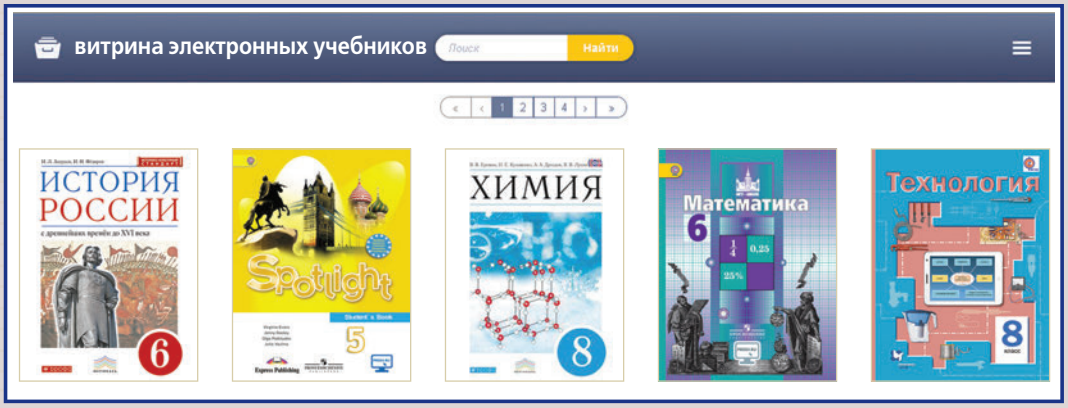
Подключайтесь!

Каждому учителю:

- предметно-методические материалы
- модульные курсы повышения квалификации
- методическая литература

Новое:

- более 100 вебинаров и видеолекций
- электронные учебники



Подробная информация и регистрация на сайте:

digital.1september.ru

Участие в проекте общеобразовательной организации и педагогических работников удостоверяется соответствующими документами.

ИСТОРИЯ ИНФОРМАТИКИ

“Учить умноженье — сплошное мученье…”

В.В. Шилов,
Москва

Надоело мне в цифрах копаться,
Заболела от них голова.
Я хотел бы забыть, что такое 17,
Что такое 4 и 2.

Николай Олейников
“Жалоба математика”

Английский монах Альдгельм Мальмсберийский (639–709), принадлежавший к числу образованнейших людей своего века, в одном из писем как-то пожаловался, что наука о вычислениях “приводит его в отчаяние”, и что только после многих трудов он “познал самый сложный из законов природы, … называемый дробями”. Что же, даже среди интеллектуальной элиты той эпохи знание самых элементарных сведений из области арифметики и в самом деле было весьма редким. А уж о простых школярах и говорить не приходится!

Впрочем, не стоит этому удивляться. Во времена Альдгельма, да и еще несколько последующих столетий, в Европе пользовались римской системой счисления, для вычислений совсем не подходящей. Но если со сложением и вычитанием люди еще кое-как справлялись, то умножение и особенно деление очень долго оставалось для европейских математиков камнем преткновения. Можно вспомнить слова выдающегося английского математика и логика Альфреда Н. Уайтхеда (1861–1947): “До появления арабских цифр умножение представляло трудность, а деление целых чисел требовало напряжения наивысших математических способностей”. Действительно, эффективное выполнение этих операций возможно только при использовании позиционной системы счисления, о которой в Европе узнали только на рубеже первого и второго тысячелетий.

Время тогда двигалось медленно, одно столетие почти ничем не отличалось от другого… Другой монах — живший спустя два с половиной века после Альдгельма француз Герберт Аврилакский (ок. 940–1003) — по достоинству считается крупнейшим ученым X века. Его необычайная ученость вызывала благоговейный страх не только у современников, но и отделенных от него несколькими поколениями потомков, — Герберта считали колдуном и чернокнижником и даже обвиняли в связи с дьяволом. Вокруг его имени возникли многочисленные легенды, ставшие впоследствии одним из источников рассказов о докторе Фаусте. Особенное изумление вызывало умение Герберта быстро делить самые большие числа — понятно, что без помощи нечистой силы здесь обойтись никак не могло… На самом же деле арифметические операции Герберт выполнял при помощи изобретенной им счетной доски (*абака*) — отдаленного “родственника” русских счетов. Для представления чисел на абаке выкладывались специальные жетоны, на которых были изображены цифры от 1 до 9 (правда, их форма еще сильно отличалась от современной). Именно благодаря Герберту и его жетонам Европа впервые узнала о десятичной системе счисления и арабских цифрах, и своими трудами он подготовил их последующее победное шествие по континенту.

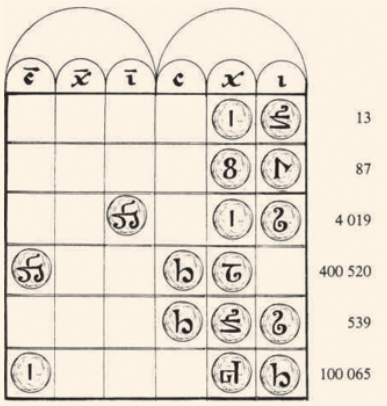


Схема абака Герберта

Однако и это изобретение не слишком помогло бедным школярам в постижении науки арифметики… Правила вычисления на абаке Герберта

были весьма сложны и столь запутанны, что даже через сто пятьдесят лет английский хронист Вильям Мальмсберийский (ок. 1090–1143) писал, что они “едва понятны трудолюбивым вычислителям”.

Так и шла многовековая борьба за овладение премудростями науки арифметики, борьба для многих совершенно безнадежная… Не случайно английские школяры XVI века жаловались, что

Учить умноженье —
Сплошное мученье,
С делением — просто беда…
Мне смысла дробей
Понять — хоть убей —
Боюсь, не дано никогда![1].

И в самом деле, еще в середине XVI века умением делить владели далеко не все ученые, а в Германии, например, искусству деления обучали лишь в одном университете — в г. Альтдорф, что близ Нюрнберга. Дневники англичанина Сэмюэла Пеписа (1633–1703) считаются у него на родине одним из наиболее интересных и ярких свидетельств о повседневной жизни людей второй половины XVII века. 4 июля 1662 г. он записал: “…я пытаюсь изучить математику (прежде всего стараюсь выучить таблицу умножения)”. Появилась эта запись не случайно, поскольку как раз в это время Пепис получил место клерка в Адмиралтействе и его работа была связана с различными подсчетами. Не стоит думать, что Пепис был невеждой. Напротив, он был человеком образованным и культурным, а впоследствии в течение нескольких лет являлся даже Президентом Лондонского Королевского общества. Скорее, тот факт, что в тридцатилетнем возрасте, имея за плечами Кембриджский университет, Пепис не умел ни умножать, ни делить, лучше всяких слов характеризует уровень тогдашнего преподавания математики.

Герберт Аврилакский и его ученики

Но и в других странах ситуация со знанием арифметики обстояла далеко не блестяще, и Россия здесь исключением не была. Английский инженер капитан Джон Перри (1670–1732), работавший в России с 1698 по 1712 г. по приглашению Петра I, после возвращения домой издал в Лондоне книгу “Состояние России при нынешнем царе…”. В ней он писал, что в стране в то время “…не существовало никакой школы, где бы преподавалась Арифметика и им не известно было употребление цифр (кажется, во всей стране не было и 20 человек, имевших об этом понятие). <…> Только весьма немногие лица умеют считать цифрами, и между собратьями своими они считаются весьма искусными людьми”.

Давайте также вспомним классическую комедию русского драматурга Дениса Ивановича Фонвизина (1745–1792) “Недоросль”, написанную в 1782 г. В ней ярко показаны некоторые особенности процесса обучения детей провинциального дворянства — в том числе обучения арифметике. Один из персонажей комедии, отставной солдат Цыфиркин, говорит о себе: “Малу толику арихметике маракую, так питаюсь… у счетных дел. Не всякому открыл господь науку: так кто сам не смыслит, меня нанимает то счетец поверить, то итоги подвести. Тем и питаюсь: праздно жить не люблю. На досуге ребят обучаю”.

Цыфиркин — один из представителей многочисленного племени учителей арифметики (в Европе их называли *мастерами счета*), которые на протяжении столетий оказывали населению именно те услуги, о которых говорит отставной солдат. Мы не знаем, сколь успешно Цыфиркин учил других ребят, но с недорослем Митрофанушкой дело у него шло из рук вон плохо: “третий год над ломаными [то есть над дробями. — **В.В. Шилов**] бьемся, да что-то плохо клеятся”; “бьюсь с ним третий год: трех перечесть не умеет”.

Итак, за три года обучения Митрофанушка не только не постиг дробей, но даже не научился считать до трех! А уж о более сложных материях зачастую и вовсе не приходилось говорить…

В рассказе графа Евгения Андреевича Салиаса (1840–1905) “Машкерад”, напечатанном в 1900 г., события происходят в самом конце XVIII века в той же среде, что и в пьесе Фонвизина. И хотя кажется, что обучение одной из героинь повествования дало несколько больший эффект, нежели у Митрофанушки, все-таки: “…Оленька далее чтения и письма не пошла. В арифметике она одолела четыре правила и остановилась перед дробями, как пред какими чудищами, которые наводили на нее робость. Да, кроме того, и голова часто стала болеть. Нянюшка ее, Власьевна, прямо заявила барину, что он ответ перед Богом отдаст за терзания своего единственного ребенка бесовскими выдумками. А уж уморить-то ее — он непременно уморит”.

А в замечательной книге “Очерки бурсы” русского писателя Николая Герасимовича Помяловско-

[1] Перевод В.В. Шилова. — **Прим. ред.**

го (1835–1863) мы находим колоритное описание урока арифметики в этом начальном духовном учебном заведении:

— Березин, скажи, на котором месте стоят десятки?

— На десятом.

— И отлично. А сколько тебе лет?

— Двадцать с годом.

— А сколько времени ты учишься?

— Девятый год.

— И видно, что ты не без успеха учился восемь лет. И вперед старайся так же.

Книга Помяловского увидела свет в 1863 г., а рассказывал в ней автор о событиях, происходивших в пору его детства — лет за двадцать до этого. Как мы видим, идея позиционной записи чисел, которая, как полагал Уайтхед, значительно упростила жизнь математикам, была все-таки чересчур сложна для усвоения ее отдельными великовозрастными бурсаками…

Не стоит думать, что описанные выше проблемы были характерны только для Старого Света. В повести Марка Твена (1835–1910) “Приключения Гекльберри Финна” главный герой не столько жалуется, сколько попросту констатирует: “Я почти что каждый день ходил в школу, научился складывать слова, читать и писать немножко и выучил таблицу умножения наизусть до шестью семь — тридцать пять, а дальше, я так думаю, мне нипочем не одолеть, хоть до ста лет учись”. Кстати, действие повести происходит в то же самое время, что и в книге Помяловского, — в начале 40-х годов XIX века.

Вспоминается фантастический рассказ американского писателя Айзека Азимова (1920–1992) “Сколько будет 9 × 7?” (другое название — “Чувство силы”), впервые напечатанный в 1958 г. В нем действие происходит в отдаленном будущем, в котором за людей все делают компьютеры, а сами люди за ненадобностью полностью забыли даже элементарную математику, и искусство счета в том числе. И вот в этом компьютеризированном мире неприметный техник низшего разряда Майрон Аюб вдруг открывает способ умножать числа без использования компьютера, на бумаге! Чтобы проверить столь невероятное сообщение, собирается комиссия из высших военных и гражданских чинов:

— …Генерал, будьте любезны задать число!

— Семнадцать, — сказал генерал.

— А вы, конгрессмен?

— Двадцать три.

— Хорошо! Аюб, перемножьте эти числа и покажите господам, как вы это делаете.

— Да, Программист, — сказал Аюб, втянув голову в плечи. Из одного кармана он извлек блокнотик, из другого — тонкое самопишущее перо и, наморщив лоб, начал тщательно выводить на бумаге значки.

<…> Аюб вновь принялся за работу, руки у него слегка дрожали. Наконец он тихо произнес:

— Ответ — триста девяносто один.

Конгрессмен Брант снова достал свой карманный компьютер и подвигал рычажками.

— Черт возьми, верно! Как он угадал?

— Это не догадка, господа, — возразил Шуман. — Он посчитал результат. И сделал это на листке бумаги.

— Чепуха, — нетерпеливо произнес генерал.

Вряд ли имеет смысл пересказывать дальнейшие события рассказа — достаточно заметить, что, как всегда, это великое открытие в первую очередь начинают использовать в военных целях.

В середине XX столетия, когда Азимов писал свой рассказ, казалось, что он сгущает краски, и что многовековые проблемы с обучением арифметике остались в далеком прошлом. Во всяком случае, уж таблицу-то умножения назубок знали даже самые закоренелые двоечники. Но не зря философы учат, что развитие идет по спирали. Похоже, что именно так дело и обстоит — увы, сегодня все чаще можно встретить школьников и даже студентов, которые таблицу умножения знают на уровне Гека Финна, а без помощи калькулятора не могут перемножить или разделить пару чисел! Именно не могут, потому что не умеют, а не ленятся… И все-таки очень не хочется, чтобы мрачное пророчество фантаста сбылось!

От редакции. Найдите ошибку в ответе Гека Финна.

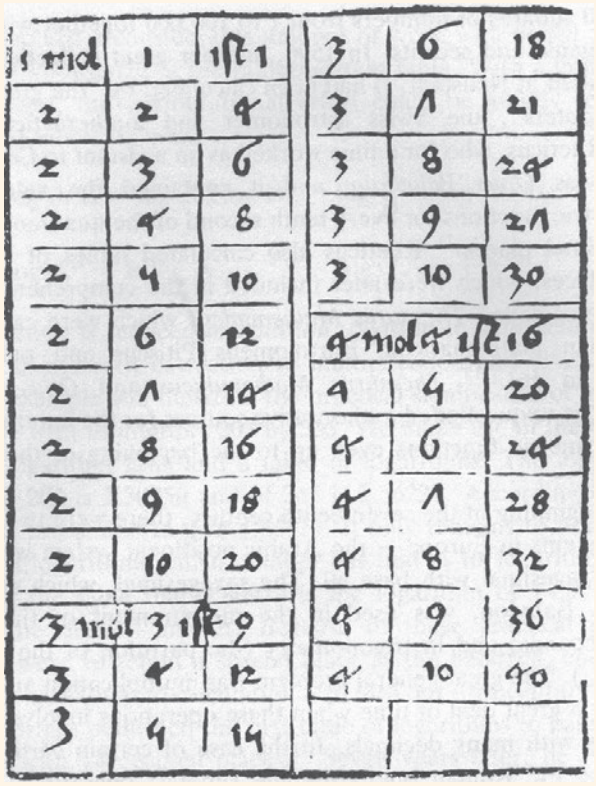


Таблица умножения, XV век

Исполнитель Множик

В системе команд исполнителя Множик имеются всего две команды, которым присвоены номера:

1) “Умножь на 2” — команда 1;

2) “Умножь на 3” — команда 2.

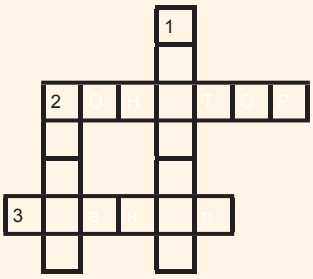
Первая увеличивает число на экране в два раза, вторая — утраивает его. Программа для Множика — это последовательность команд. Сколько различных чисел можно получить из числа 13 при помощи программы, содержащей ровно шесть команд?

Сколько значащих нулей?

Сколько значащих нулей в двоичной записи шестнадцатеричного числа $13AC,E8_{16}$? Ноль называется значащим, если его удаление из записи числа ведет к изменению значения числа.

Кроссворд без комментариев

О приведенном ниже кроссворде известно только то, что в нем должны быть записаны названия устройств ввода и вывода информации в персональном компьютере. Решите, пожалуйста, его.



Из одной последовательности — другая. Вариант 2

Из последовательности $x_1, x_2, \ldots, x_n$, в которой x_i может быть равно 0 или 1, получена последовательность $y_1, y_2, \ldots, y_{n-1}$ по следующему правилу:

$$y_i = x_i + x_{i+1}, i = 1, 2, \ldots n - 1.$$

Определите, какие подпоследовательности (части последовательности) значений y_i не могут быть получены указанным способом. Обратим внимание, что если, например, не может быть получена подпоследовательность $a\ b$, то подпоследовательности $a\ b\ c$, $c\ d\ a\ b$, $c\ d\ d\ a\ b$ и т.п. не являются ответами, поскольку все они уже включают фрагмент $a\ b$.

Для выявления особенностей формирования последовательности y целесообразно использовать электронную таблицу Microsoft Excel или подобную программу.

	A	B	C	…	L	M
1	x	1	0		1	1
2	y	1			2	
3						

В строке 1 значения 0 и 1 целесообразно не вводить вручную, а использовать функцию СЛЧИС, которая будет возвращать случайное целое число, принимающее значение 0 или 1 (понадобится также функция ЦЕЛОЕ). В этом случае при каждом нажатии функциональной клавиши F9 в строке 1 будут выводиться новые значения.

Формулу для расчета, по которой будут возвращаться значения 0 или 1, установите самостоятельно. Естественно, значения y также должны рассчитываться по формулам.

Примечание. В названии задачи указан “Вариант 2”, так как в предыдущем выпуске “В мир информатики” была опубликована похожая задача. Там же были опубликованы соответствующие программы для получения последовательности y.

Числовой ребус с РЕПКОЙ ☺

Решите, пожалуйста, числовой ребус:

ДЕДКА + БАБКА + РЕПКА = СКАЗКА

Найдите все решения.

ПОИСК ИНФОРМАЦИИ

Четыре вопроса

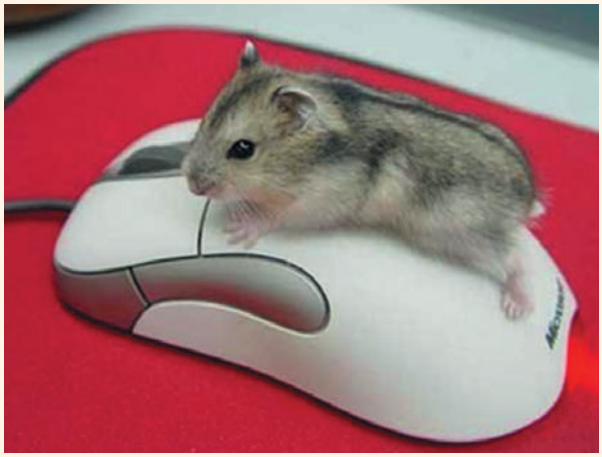
1. Какой богине стоило молиться забывчивому древнему греку?

2. Какая разведка затевает интригу в “Шпионском романе” писателя Бориса Акунина?

3. Какую куклу запрещала покупать своим дочерям мать актрисы Николь Кидман?

4. Какой аппарат ныне используют для облегчения семян ради многократного увеличения урожая?

ПРИ1Ы



ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным в разделе “В мир информатики” ранее

Задача “Об исполнителе Черепашке, или Готовимся к ОГЭ”

Напомним, что были предложены три задания.

1. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 7 [Вперед 100 Направо 60 Вперед 20]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) правильный треугольник;
3) правильный семиугольник;
4) правильный шестиугольник.

2. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 14 [Направо 120 Вперед 50
           Налево 90]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) правильный квадрат;
3) правильный двенадцатиугольник;
4) правильный треугольник.

3. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 9 [Вперед 15 Направо 70]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) правильный двадцатичетырехугольник;
3) правильный девятиугольник;
4) правильный пятиугольник.

Учитывая, что задания, аналогичные предложенным, встречаются в заданиях ОГЭ и ЕГЭ по информатике, обсудим решения подробно.

Задание 1

Анализ (и практика построения приведенного алгоритма) показывает, что при наличии в теле цикла (в квадратных скобках) двух команд `Вперед 100` и `Вперед 20`, между которыми имеется одна или несколько команд `Направо`, в результате будет нарисована линия длиной 100 + 20 = = 120. Аналогично, если используется команда `Налево`. Это важный вывод, который следует учитывать при решении задач такого типа. Количество таких линий будет равно числу повторений тела цикла (числу в команде `Повтори`). Осталось определить, какую фигуру образуют эти линии.

В статье было показано, что в алгоритмах типа:

```
Повтори n [Вперед a Направо угол]
```

(а заданный в условии алгоритм, по сути, является именно таким) — если произведение числа повторений *n* на угол поворота *угол* равно 360, то будет построен правильный *n*-угольник. Примеры сочетаний значений *n* и *угол* приведены в таблице:

n	*угол*
3	120
4	90
5	72
6	60
9	40
12	30
…	…

Если значение *n* в алгоритме задано большим, чем указано в таблице, то исполнитель после рисования правильного *n*-угольника продолжит перемещаться по уже нарисованным линиям ☹, если же меньшим — то будет нарисована незамкнутая ломаная линия ☹.

Учитывая это, можно сделать вывод, что в рассматриваемом случае будет нарисован правильный шестиугольник (ответ 4).

Задание 2

Здесь важно разобраться, что происходит, когда в теле цикла есть две команды поворота, между которыми имеется одна или несколько команд `Вперед`. Анализ показывает, что в результате выполнения команд `Направо 120 Вперед 50 Налево 90` будет нарисована линия длиной 50 и Черепашка отклонится от исходного положения на 120 – 90 = = 30 градусов вправо. Затем все повторится 14 раз. Так как 14 больше, чем 12 (число повторения для угла 30 — см. таблицу), то это значит, что будет нарисован правильный двенадцатиугольник (хотя некоторые действия исполнителя будут излишними).

Правильный вариант ответа — 3.

В общем случае при анализе алгоритмов, аналогичных приведенному в условии, следует учитывать направление поворотов исполнителя и определить “результирующий” угол поворота после однократного выполнения тела цикла. При превышении 360 градусов необходимо определить остаток от деления величины “результирующего” угла на 360. Для определения вида полученной фигуры нужно сравнить заданное число повторений тела цикла, “результирующий” угол, число повторений *n* и соответствующий угол из приведенной выше таблицы.

Задание 3

Здесь угол поворота такой (70), что он не является делителем числа 360, то есть произведение угла поворота на число повторений тела цикла не может быть равно 360. Это говорит о том, что правильный многоугольник не будет построен.

Правильный вариант ответа — 1.

Правильные ответы представили:

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**;

— Никаноров Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

№ пп	Алгоритм	Ответ			
		1	*2*	*3*	*4*
1	`Повтори 8 [Вперед 15 Направо 45]`				
2	`Повтори 7 [Вперед 25 Налево 35]`				
3	`Повтори 10 [Вперед 15 Направо 30]`				
4	`Повтори 18 [Вперед 15 Вперед 10 Налево 20]`				
5	`Повтори 12 [Вперед 20 Направо 60]`				
6	`Повтори 6 [Направо 90 Вперед 50 Налево 30]`				
7	`Повтори 10 [Вперед 15 Налево 30 Вперед 10]`				
8	`Повтори 20 [Направо 15 Налево 30 Вперед 50]`				
9	`Повтори 6 [Направо 60 Вперед 50 Налево 90]`				
10	`Повтори 12 [Направо 180 Вперед 50 Направо 60 Направо 180]`				

— Стороженко Степан, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Хорькова Анна, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Новые задания для самостоятельной работы

1. Заполните полностью таблицу, приведенную выше при анализе.

2. Ответьте на вопрос: “Может ли угол поворота в этой таблице быть нецелым?”.

3. Заполните приведенную выше таблицу, указав с помощью символа “+” вид фигуры, которая будет получена в результате выполнения исполнителем Черепашка того или иного алгоритма.

Возможные ответы:

1 — правильный многоугольник без лишних операций;

2 — правильный многоугольник с лишними операциями;

3 — “недостроенный” правильный многоугольник;

4 — другая незамкнутая ломаная линия.

(См. таблицу вверху страницы.)

Числовой ребус “КАПЛЯ + КАПЛЯ + КАПЛЯ = ОЗЕРКО”

Решение ребуса, записанного “в столбик”:

```
    6 7 9 5 4
+   6 7 9 5 4
    6 7 9 5 4
-------------
2 0 3 8 6 2
```

Обратим внимание на то, что в ряде ответов одна и та же буква имеет разные значения, что недопустимо в головоломках такого типа.

Решение головоломок “Два квадрата” и “Три квадрата” представили также:

— Ермолов Арсений, Хозин Марат и Чуб Алексей, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Иголкина Елена, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**

Все перечисленные читатели будут награждены дипломами. Молодцы!

Правильные ответы на задачи “Кто на каком инструменте играет?”, “Постройка плотины” и “Есть ли такая цифра?” и решение японских головоломок судоку, опубликованных в сентябрьском выпуске “В мир информатики”, представила также Маслова Арина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**

Задачу “Постройка плотины” правильно решил также Мягков Александр, ученик 5-го класса Сапожковской средней школы имени Героя России Тучина Алексея Ивановича, Рязанская обл., учитель **Задубровский В.Ф.**

Задача “Три одноклассника”

Напомним условие: “Три одноклассника — Влад, Тимур и Юра, встретились спустя 10 лет после окончания школы. Выяснилось, что один из них стал врачом, другой — физиком, а третий — юристом. Один полюбил туризм, другой — бег, страсть третьего — регби. Юра сказал, что на туризм ему не хватает времени, хотя его сестра — единственный врач в семье, заядлый турист. Врач сказал, что он разделяет увлечение коллеги. Забавно, но у двоих из друзей в названиях их профессий и увлечений не встречается ни одна буква их имен. Определите, кто чем любит заниматься в свободное время и у кого какая профессия”.

Решение

Из слов Юры ясно, что он не увлекается туризмом и он не врач. Из слов врача следует, что он турист. Следовательно, можно составить такую таблицу:

Имя	Юра		
Профессия		Врач	
Увлечение		Туризм	

Буква “а”, присутствующая в слове “врач”, указывает на то, что Влад тоже не врач, следовательно, врач — Тимур. В его имени есть буквы “т” и “р”, встречающиеся в слове “туризм”, следовательно, второй из друзей, в названиях

профессии и увлечения которого не встречается ни одна буква его имени, — Юра. Юра не юрист и не регбист, так как в его имени содержатся буквы “ю” и “р”. Следовательно, окончательно имеем:

Имя	Юра	Тимур	Влад
Профессия	Физик	Врач	Юрист
Увлечение	Бег	Туризм	Регби

Ответ: Влад — юрист и регбист, Тимур — врач и турист, Юра — физик и бегун.

Правильный ответ прислали:

— Абдрахманова Валерия, Задорина Наталья, Межогских Дарья, Орлов Кирилл и Салимов Владислав, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Абдувахидова Алина, Абдувахидова Софья и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Барановская Татьяна, Голубь Иван и Макаркина Светлана, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гулевских Анна, Бражникова Ксения и Воронков Андрей, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Густова Дарья, г. Ярославль, гимназия № 1, учитель **Тишининова Я.Н.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Донникова Анна и Потапенко Михаил, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Казанец Елена и Ломтева Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**;

— Суховеева Елена и Хрущ Анна, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Тарасова Наталья, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Отметим ответы учащихся школы № 124 г. Челябинска, приведших подробное обоснование ответа.

Головоломка “Переложить спичку”

Напомним, что следовало в неправильном неравенстве, выполненном из спичек, — 8 – 6 = 0, переложить одну спичку так, чтобы оно стало верным.

Решение

Нужно сделать так, чтобы получилось: 6 – 6 = 0.

Правильное решение предложили:

— Абакумов Кирилл, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Абдувахидова Алина, Абдувахидова Софья, Матрёнина Екатерина, Милушкин Дмитрий, Фролова Александра и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Бакай Максим, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Балаева Любовь, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Лагоша Артем, Республика Коми, г. Усинск, Центр дополнительного образования детей, “Школа программистов”, педагог **Казанцева О.В.**;

— Мазаев Тарас и Макаркина Ольга, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**;

— Никонов Петр и Пулин Андрей, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Хрущ Анна, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Шаронова Наталья, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

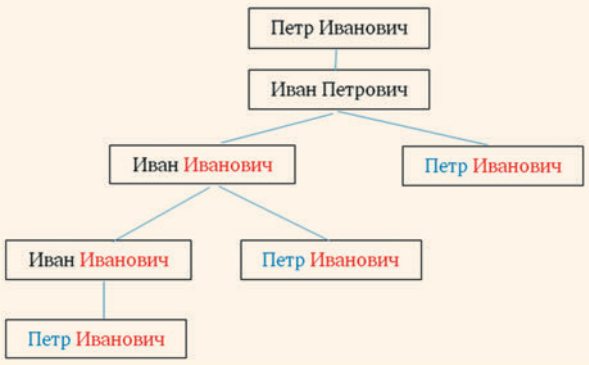
Задача “Потомки”

Напомним условие: “Среди потомков Петра Ивановича по мужской линии (сыновья, сыновья сыновей и т.д.) ровно три Петра и пять Ивановичей. При каком наименьшем числе потомков это возможно? (Имена двух любых братьев различны.)”.

Решение

Чтобы в роду появился Иванович, сначала должен появиться Иван. Первый Иван не будет Ивановичем, поэтому пять Ивановичей не совпадут с первым Ивановичем. Значит, потомков должно быть не менее шести. Покажем, как можно обойтись шестью потомками.

У Петра Ивановича сын Иван; у Ивана два сына: Иван II и Петр, у Ивана II два сына: Иван III и Петр, у Ивана III сын Петр.



Правильный ответ представили:

— Абдувахидова Алина, Абдувахидова Софья и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Бакай Максим и Деминцев Борис, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Бражникова Ксения, Гулевских Анна и Воронков Андрей, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Ермолин Александр, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Лагоша Артем, Республика Коми, г. Усинск, Центр дополнительного образования детей, "Школа программистов", педагог **Казанцева О.В.**;

— Платонов Иван, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Старостин Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**

Головоломка "Русские числительные"

Напомним, что следовало определить количество двузначных чисел, которые записываются двумя словами, начинающимися на одну и ту же букву.

Решение

Прежде всего подходят все двузначные числа с одинаковыми цифрами десятков и единиц, кроме 44 и 11 (оно вообще записывается одним словом). Таких чисел 7. Кроме того, находим числа 29, 92 и 47.

Ответ: 10 чисел.

Правильный ответ представили:

— Абакумова Алевтина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Викторов Олег, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гаронин Лев, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Густова Дарья, г. Ярославль, гимназия № 1, учитель **Тишининова Я.Н.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**;

— Микаелян Артем, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Милушкин Дмитрий и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Платонов Иван, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**

Задача "Поэт-врун"

Напомним условие: "Поэт Ляпис-Трубецкой говорит правду с полуночи до полудня и лжет в остальное время суток. Ежедневно он сочиняет стихи с 11-00 до 15-00. Сколько часов в сутках, когда он может заявлять: «Сейчас я сочиняю стихи!»?".

Решение

"Герой" задачи с полуночи до полудня говорит правду, поэтому сказать: "Сейчас я сочиняю стихи!" — он может тогда, когда он их действительно сочиняет, — с 11-00 до 12-00 (1 час). С полудня до полуночи он лжет, поэтому сказать: "Сейчас я сочиняю стихи!" — он может только тогда, когда он их не сочиняет, а именно — с 15-00 до 24-00 (9 часов). Всего получается 10 часов.

Ответ: 10 часов.

Правильный ответ прислали:

— Абакумова Алевтина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Бражникова Ксения, Гулевских Анна и Воронков Андрей, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Викторов Олег, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Густова Дарья, г. Ярославль, гимназия № 1, учитель **Тишининова Я.Н.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Ермолин Александр, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Микаелян Артем, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Милушкин Дмитрий и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Никитин Тимофей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Головоломка "Кубик"

Напомним условие: "На гранях кубика записано по одной цифре от 1 до 6. Если на этот кубик смотреть с одной стороны, то видны 1, 2 и 3, с другой стороны видны 4, 1 и 5, а с третьей стороны видны 2, 6 и 4. Какие цифры находятся с противоположных сторон цифр 1 и 2?"

Решение

Вид кубика с первой из указанных в условии сторон показан на *рис*. 1. При этом размещение цифр на гранях может быть разным. Но в любом случае на двух гранях рядом с цифрой могут быть цифры 6 и 4 (третья сторона из условия). Значит, оставшаяся цифра 5 — с противоположной от двойки стороны.

Рис. 1

Осталось узнать, где 4 и где 6. По условию, рядом с единицей — цифра 5 (на *рис*. 1 — снизу) и цифра 4 (напротив тройки). Значит, цифра 6 — напротив единицы. Развертка кубика показана на *рис*. 2 (ее прислал Артем Лагоша, Республика Коми, г. Усинск, Центр дополнительного образования детей, “Школа программистов”, педагог **Казанцева О.В.**).

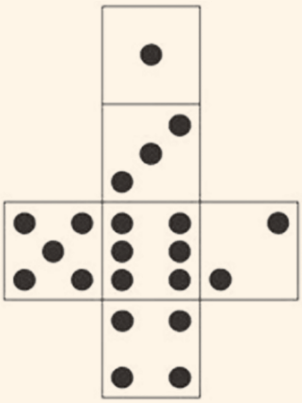


Рис. 2

Все сделанные рассуждения, как справедливо указала в своем ответе Дарья Межогских, г. Челябинск, школа № 124 (учитель **Юртаева Г.Ю.**), можно сопроводить рисованием цифр на гранях, если изобразить невидимые на *рис*. 1 грани кубика.

Очень логичное и понятное объяснение ответа привел Кирилл Орлов из школы № 124 г. Челябинска: “Цифра 1 в условии была вместе с цифрами 2, 3, 4 и 5. Значит, на противоположной стороне — 6. Цифра 2 в условии была вместе с цифрами 1, 3, 4 и 6. Следовательно, на противоположной стороне — 5”.

Кроме перечисленных читателей, ответы прислали:

— Абакумов Кирилл и Мартыненко Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Абдрахманова Валерия, Задорина Наталья и Салимов Владислав, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Бакай Максим, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Балаева Любовь и Середа Светлана, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Бражникова Ксения, Гулевских Анна и Воронков Андрей, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Мазаев Тарас и Макаркина Ольга, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Никонов Петр и Пулин Андрей, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Чуркин Никита, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Шаронова Наталья, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Задание “Три вопроса по истории” (рубрика “Поиск информации”)

Ответы

1. Русские корабли, которым в 1904 году японская эскадра потребовала сдаться, — “Варяг” и “Кореец”.

2. Шашке, которая полагалась городовому в полиции царской России, остряки дали прозвище “селедка”.

3. Известный исторический деятель, которого вызвал на дуэль анархист Михаил Бакунин за нелестный отзыв о качествах русских солдат, — Карл Маркс.

Ответы прислали:

— Абаева Алина, Коновалова Светлана и Юркина Ирина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Абдувахидова Алина, Абдувахидова Софья и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Барановская Татьяна и Жукова Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Блюденов Кирилл, Гулевских Анна, Долгова Анна, Загороднев Александр, Кокорева Виктория, Недикова Алиса, Обухова Элина, Раджабова Алина и Утробина Ирина, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Губарева Инна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Дикань Николай и Заикина Татьяна, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Елисеева Мария, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Ивакина Лина, Калинина Екатерина, Князев Антон и Чунин Павел, средняя школа г. Пионерский Калининградской обл., учитель **Багрова О.А.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Макаров Игорь и Никанорова Инна, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**

Заметим, что приведенный в ряде ответов на третий вопрос Катков действительно был вызван на дуэль Бакуниным, но не в связи с причиной, указанной в вопросе. К тому же русского публициста и литературного критика М.Н. Каткова трудно назвать “известным историческим деятелем”.

Задача “На новогоднем балу”

Условие: “Как-то раз четыре товарища (Петя, Боря, Леша и Коля) пошли со своими сестрами (Светой, Наташей, Олей и Леной) на школьный новогодний бал.

Определите, кто чей брат и кто с кем танцевал во время первого танца, если:

а) каждый из ребят не танцевал со своей сестрой;

б) Лена танцевала с Петей, а Светлана — с братом Наташи;

в) Оля танцевала с братом Светланы, Боря — с сестрой Алеши, а Алеша — с сестрой Пети”.

Ответ

Петя — брат Оли — танцевал с Леной, Боря — брат Лены — танцевал с Наташей, Алеша — брат Светы — танцевал с Олей, Коля — брат Наташи — танцевал со Светой.

Правильные ответы представили:

— Абдрахманова Валерия, Задорина Наталья, Межогских Дарья, Орлов Кирилл и Салимов Владислав, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Абдувахидова Алина и Абдувахидова Софья, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Барановская Татьяна, Зорькина Мария и Макаркина Светлана, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Василенко Елена и Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Густова Дарья, г. Ярославль, гимназия № 1, учитель **Тишининова Я.Н.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Донникова Анна и Потапенко Михаил, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Казанец Елена и Ломтева Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Суховеева Елена и Хрущ Анна, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Тарасова Наталья, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Задача “В зоопарке”

Напомним условие: “В зоопарке на трех закрытых со всех сторон клетках, в двух из которых находились тигр и леопард, висели таблички с надписями:

1) на первой клетке: “Здесь сидит тигр”;

2) на второй клетке: “Здесь находится леопард”;

3) на третьей клетке: “Тигр сидит во второй клетке”.

Работник зоопарка заявил, что только одно из этих утверждений является истинным. Не ошибся ли он?”.

Решение

В каком случае работник зоопарка (в одном из ответов, полученных редакцией, он назван “дворником” ☺) ошибся? В том случае, если нет такого варианта размещения животных, при котором одно из утверждений на табличках истинно и два — ложны.

Всего возможны шесть вариантов размещения двух животных в трех клетках:

№	Первая клетка	Вторая клетка	Третья клетка
1	Тигр	Леопард	
2	Леопард	Тигр	
3		Тигр	Леопард
4		Леопард	Тигр
5	Тигр		Леопард
6	Леопард		Тигр

Можно проверить каждый из этих вариантов и утверждения на табличках на истинность. Например, для 1-го варианта размещения:

	Первая клетка	Вторая клетка	Третья клетка
	Тигр	Леопард	
Утверждение	“Здесь сидит тигр”	“Здесь находится леопард”	“Тигр сидит во второй клетке”
	Истинно	Истинно	Ложно

Для этого варианта слова работника зоопарка ошибочны, так как истинными являются два утверждения на табличках.

Но надо проверить остальные варианты. Для варианта № 6 результаты анализа такие:

	Первая клетка	Вторая клетка	Третья клетка
	Леопард		Тигр
Утверждение	“Здесь сидит тигр”	“Здесь находится леопард”	“Тигр сидит во второй клетке”
	Ложно	Ложно	Ложно

Здесь также можно сказать, что не исключено, что работник зоопарка ошибся.

Предлагаем читателям проверить остальные варианты.

Можно также сделать те или иные допущения. Пронумеруем утверждения, записанные на табличках, соответственно, № 1, 2 и 3.

Предположим, что истинным является только утверждение № 1 (в первой клетке сидит тигр). Тогда утверждение № 3 — ложное. Это позволит определить, где находится леопард. Проверьте этот и два других варианта истинности одного утверждения.

Ответы представили:

— Абакумова Алевтина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Абдрахманова Валерия, Задорина Наталья, Межогских Дарья и Салимов Владислав, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Абдувахидова Алина, Абдувахидова Софья и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Бойко Николай и Мезенцев Владимир, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Бражникова Ксения, Гулевских Анна и Воронков Андрей, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Великородных Константин, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Гришин Антон, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Кондауров Максим, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Ломтев Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Заметим, что в ряде полученных редакцией писем правильный ответ не был подтвержден соответствующим размещением животных в той или иной клетке либо обоснован не всеми возможными вариантами размещения.

Задача “Мамед и Самед”

Напомним условие: “Летом Мамед и Самед очень любили есть вишню и черешню. Однажды мама купила бидон вишни (208 штук) и ведерко черешни (320 штук).

Известно, что Мамед ест одну вишенку за 5 секунд, а одну черешенку — за 8 секунд. Самед же одну вишенку ест за 6 секунд, а одну черешенку — за 4 секунды.

1. Какое минимальное время понадобится каждому брату на поедание ягод, если они разделят поровну и вишни, и черешни?

2. За какое минимальное время мальчики могут съесть все эти ягоды?

Брать в рот и есть сразу несколько ягод — неприлично и опасно ☺”.

Решение

1. Мамед съест свою половину вишен (104 шт.) за $104 \times 5 = 520$ секунд и свою половину черешен (160 шт.) за $160 \cdot 8 = 1280$ секунд, что всего составит 1800 секунд, или 30 минут. Самед свою долю вишен съест за 624 секунды и свою половину черешен за 640 секунд, а всего за 1264 секунды.

2. Чтобы съесть все за минимальное время, нужно, чтобы каждый начал есть то, что делает ☺ быстрее. Мамед съест все вишни за $208 \times 5 = 1040$ секунд. За это время Самед съест $1040 : 4 = 260$ черешен из 320. Останется 60 черешен.

Если за 8 секунд Мамед ест одну черешенку, а Самед — две, то вместе за это время они съедят три черешенки. Значит, на оставшиеся 60 черешен им понадобится $60 : 3 \times 8 = 160$ секунд, а всего они потратят $1040 + 160 = 1200$ секунд, или 20 минут.

Правильные ответы прислали:

— Абдувахидова Алина и Абдувахидова Софья, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Великородных Константин, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Гришин Антон, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Кондауров Максим, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Ломтев Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Мезенцев Владимир, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Платонов Иван, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Старостин Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**

Новая задача. “Винни-Пух и другие”

Как-то Кролик торопился на встречу с осликом Иа-Иа, но к нему неожиданно пришли Винни-Пух и Пятачок. Будучи хорошо воспитанным, Кролик предложил гостям подкрепиться. Пух завязал салфеткой рот Пятачку и в одиночку съел 10 горшков меда и 22 банки сгущенного молока, причем горшок меда он съедал за две минуты, а банку молока — за минуту. Узнав, что больше ничего сладкого в доме нет, Пух попрощался и увел Пятачка. Кролик с огорчением подумал, что он бы не опоздал на встречу с осликом, если бы Пух поделился с Пятачком. Зная, что Пятачок съедает горшок меда за пять минут, а банку молока за три минуты, Кролик вычислил наименьшее время, за которое гости смогли бы уничтожить его запасы.

Чему равно это время? (Банку молока и горшок меда можно делить на любые части.)

Источник: www.diofant.ru

Японские головоломки судоку, опубликованные в ноябрьском выпуске, решили:

— Багрова Анастасия, средняя школа г. Пионерский Калининградской обл., учитель **Багрова О.А.**;

— Гришин Антон, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Кондауров Максим и Тимофеев Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Лагоша Артем, Республика Коми, г. Усинск, Центр дополнительного образования детей, "Школа программистов", педагог **Казанцева О.В.**;

— Ломтев Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Медведева Анастасия, Владимирская обл., г. Струнино, школа № 11, учитель **Волкова Т.П.**;

— Мезенцев Владимир, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Милушкин Дмитрий и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Платонов Иван, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Старостин Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**

Кто какие фигуры вырезал?

Коля, Митя, Миша, Глеб и Алеша вырезали из бумаги разные фигуры: круг из бумаги в клетку, круг из бумаги в линейку, квадрат из бумаги в клетку, квадрат из бумаги в линейку, флажок из белой бумаги. Глеб и Митя вырезали круги, Глеб и Коля вырезали из бумаги в клетку, Коля и Миша вырезали квадраты. Кто какие фигуры вырезал?

Задача предназначена для учащихся 1–7-х классов.

Итоги конкурса № 113

Напомним, что в качестве заданий этого конкурса следовало выполнить задания, предложенные для самостоятельной работы в статьях "Еще раз о «счастливых» билетах" (рубрика "Школа программирования") и "Считаем «счастливые» билеты в среде электронных таблиц".

Результаты работы представили:

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Великородных Константин, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Галочкин Александр, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Гришин Антон, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Лебедева Екатерина и Назаркина Татьяна, средняя школа г. Пионерский Калининградской обл., учитель **Багрова О.А.**;

— Ломтев Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Попов Владимир, Республика Коми, г. Усинск, Центр дополнительного образования детей, "Школа программистов", педагог **Казанцева О.В.**;

— Старостин Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Цуцков Илья, Ардатовский коммерческо-технический техникум, поселок Ардатов Нижегородской обл., преподаватель **Зудин В.П.**;

— Чурилов Даниил, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Все перечисленные читатели будут награждены дипломами. Поздравляем!

КРЕПКИЙ ОРЕШЕК

Как обычно, в этой рубрике редакция проводит разбор задач, решение которых вызвало трудности.

Задача "Чему равно АС?"

Напомним условие: "Если число **АС** больше числа **С** на 160, а число **СЕВА** больше числа **ЕВА** на 49152, то чему равно число **АС**? Одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры. Все известные числа в условии — десятичные".

Начало решения

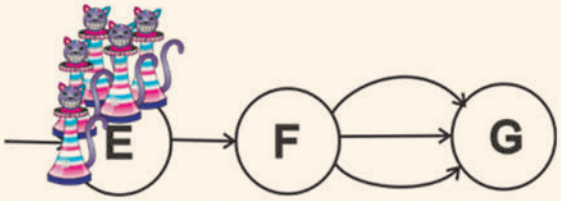
Из так называемой "развернутой формы" числа **АС** (**А**q + **С**, где q — неизвестное основание системы счисления, в которой записаны все числа) следует, что **А**q = 160. Значит, q — делитель числа 160. С другой стороны, так как **А** — это цифра, которая не может быть больше q, то можно установить, что q — одно из чисел: 16, 20, 32, 40, 80, 160. Остается проверить все перечисленные значения, учитывая разность чисел **СЕВА** и **ЕВА**, и найти подходящие значения q и **С**. Предлагаем читателям провести такие расчеты и прислать ответ в редакцию.

Задания для самостоятельной работы, предложенные в статье "Чеширский Кот путешествует по графу"

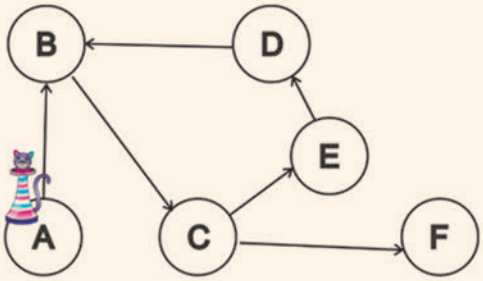
В статье описывалась методика решения задачи нахождения числа маршрутов на графе, основанная на использовании копий "котов". Обратим внимание, что задания, аналогичные предложенным, встречаются в заданиях ОГЭ и ЕГЭ по информатике, поэтому знакомство с методикой их решения, описанной в статье, является полезным.

Напомним условия задач.

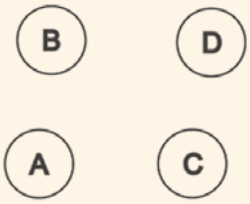
1. В пункт **E** пришло пять копий Чеширского Кота. Сколько копий придет в пункт **G**?



2. Сколько копий понадобится Чеширскому Коту, чтобы подсчитать количество маршрутов от пункта **A** до пункта **F**?



3. Нарисуйте пять дорог, соединяющих четыре пункта так, чтобы от пункта **A** до пункта **D** можно было добраться двумя способами.



4. Когда Чеширский Кот отправил из пункта **A** в пункт **H** свои копии, на дорогу опустился туман, и часть дорог стала не видна (см. *рис.* 1). Известно, что в конечный пункт пришли 15 копий.

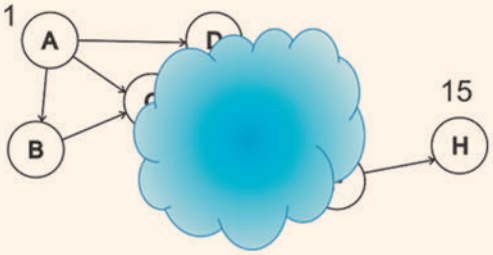


Рис. 1

Второй раз Кот отправил копию, когда перекрыли две дороги из пункта **A** так, как показано на рисунке. До пункта **H** добрались три кота.

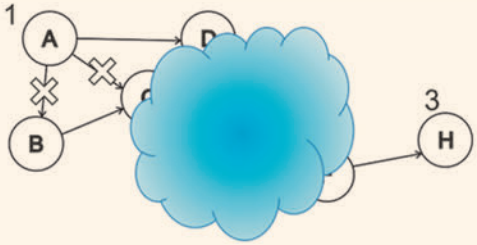


Рис. 2

Сколько котов придет в пункт **H**, когда перекроют другие две дороги (см. *рис.* 3)?

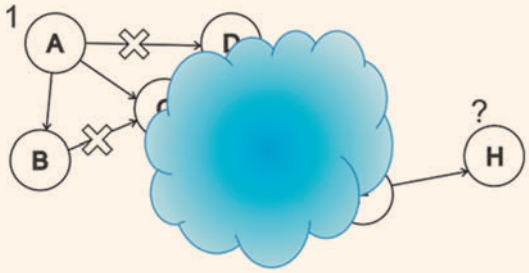


Рис. 3

Ответы и решения

Задание 1

Из пункта **E** выходит одна дорога, поэтому из него в пункт **F** выйдут те же пять копий, которые на выходе из пункта **F** “утроятся”.

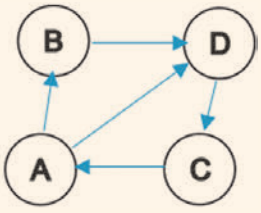
Ответ: 15.

Задание 2

Коту понадобятся две копии, но так как дорога имеет участок-петлю, возвращающий в пункт, в котором уже были (циклический участок), то подсчитать количество путей невозможно, так как всегда будет копия, отправляющаяся на участок с возвратом.

Задание 3

Один из двух возможных вариантов:



Задание 4

Это задание — самое трудное.

Так как “невидимая” часть графа имеет много возможных вариантов, перебор которых затруднен, то будем анализировать, так сказать, “обратную задачу”. Речь идет вот о чем.

Если мы знаем, что из пункта **A** в пункт **B** можно добраться пятью разными маршрутами, то каким количеством маршрутов можно попасть из пункта **B** в пункт **A**? Ответ — также пятью. Это значит, что в нашей задаче число маршрутов из пункта **A** в пункт **H** равно числу маршрутов из пункта **H** в пункт **A**. Поэтому рассмотрим задачу перемещения Чеширского Кота именно по последнему маршруту.

При такой задаче для случая на *рис.* 1 в пункт **A** придут 15 котов, а на *рис.* 2 — 3 кота, или на 12 меньше. Это уменьшение произошло из-за того, что из пункта **C** закрыли две дороги.

Предлагаем читателям продолжить рассуждения и найти ответ.

А здесь хотим сообщить, что дипломом будет награжден Хозин Марат, ученик школы № 11 г. Струнино Владимирской обл. (учитель **Волков Ю.П.**), правильно ответивший на все вопросы. Молодец, Марат!

Новая задача о Зеленом городе

Напомним условие: “В Зеленом городе 8200 домов — часть деревянных, остальные — каменные. 2500 домов стоят на правом берегу единственной речки этого города. На карту нанесли 6500 домов (все каменные дома и все дома на левом берегу реки). Что можно узнать о числе домов разного типа на каждом берегу реки?”.

Решение

Запишем известную информацию в таблицу:

	Левый берег	Правый берег	Всего
Каменные			
Деревянный			
Всего		2500	8200

Из нее следует, что на левом берегу находятся 8200 – 2500 = 5700 домов. Тогда из 6500 нанесенных на карту домов на правом берегу из камня выполнены 6500 – 5700 = 800 домов. Запишем новые данные:

	Левый берег	Правый берег	Всего
Каменные		800	
Деревянный			7400
Всего	5700	2500	8200

Можно ли определить значения в пустых клетках таблицы? Ответ на вопрос предлагаем дать читателям.

ЯПОНСКИЙ УГОЛОК

Два судоку

Решите, пожалуйста, две японские головоломки “судоку”:

1) простую:

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

2) сложную:

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

МЫСЛИ

Образование — клад, труд — ключ к нему.
Пьер Буаст, французский поэт (1765–1824)

Образование есть украшение в счастье и убежище в несчастье.
Аристотель, древнегреческий философ

ШКОЛА ПРОГРАММИРОВАНИЯ

Когда человек хочет передвинуть гору, он начинает с того, что убирает маленькие камни.

Восточная мудрость

Треугольные числа

Одинаковые шары (или кружочки — монеты и т.п.) можно размещать на плоскости так, чтобы получались различные фигуры — треугольники, квадраты, шестиугольники и т.д. Рассмотрим “упаковки” шаров в равносторонние треугольники. На *рис.* 1 изображены первые пять таких треугольников.

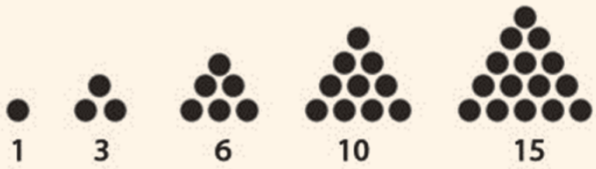


Рис. 1

Так вот, числа, соответствующие количеству шаров (кружочков), которые могут быть расставлены в форме правильного треугольника, и называются “треугольными” (в рассмотренных на *рис.* 1 случаях это числа 1, 3, 6, 10 и 15).

Разработаем программы решения ряда задач, связанных с треугольными числами. Как принято в разделе “В мир информатики”, методику разработки программ опишем с использованием школьного алгоритмического языка (система программирования КуМир). Русский синтаксис этого языка делает его максимально понятным, и вы легко сможете разработать соответствующие программы на языке программирования, который изучаете.

Задача 1 “Определить значение *k*-го треугольного числа”

Чтобы установить зависимость *k*-го треугольного числа от значения номера *k*, составим таблицу:

Номер *k*	1	2	3	4	5
Число	1	3	6	10	15

А можно ли продолжить таблицу дальше, без помощи рисунков? Сделать это совсем просто, если понять правило, по которому каждое следующее треугольное число получается из предыдущего. Посмотрите на таблицу: третье треугольное число получается, если ко второму треугольному числу прибавить число 3, то есть его номер; четвертое треугольное число получается добавлением к третьему числу 4 и т.д. Это же правило применимо и ко второму треугольному числу.

Первый вариант программы и основан на этой закономерности:

```
алг Треугольное_число_Вариант_1
нач цел k, число, номер
вывод нс, "Задайте номер числа "
ввод k
```

```
  число := 1 |Известное 1-е число
  |По установленному правилу
  |рассчитываем последующие
  |треугольные числа до k-го
  нц для номер от 2 до k
    число := число + номер
  кц
  вывод нс, k, "-е треугольное число: ",
число
  кон
```

Заметим, что формулы для определения значений ряда чисел, в которых значение каждого числа, начиная с определенного, зависит от значения одного или нескольких предыдущих чисел, а именно такой является полученная нами формула для расчета i-го треугольного числа ТЧ$_i$:

$$ТЧ_i = ТЧ_{i-1} + i,$$

— называется “рекуррентной формулой”, или “рекуррентным соотношением”.

А можно ли найти какое-нибудь треугольное число, не вычисляя всех предыдущих? Ведь нам не нужны все треугольные числа, а только k-е!

Попробуем найти треугольное число под номером 10. Согласно *рис.* 1, десятое треугольное число равно сумме:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.$$

Для подсчета этой суммы запишем ее слагаемые в обратном порядке и расположим суммы одна под другой:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.$$
$$10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1.$$

Сумма каждой пары, расположенной друг под другом, равна 11. Всего таких сумм 10. Поэтому удвоенная сумма равна 10×11. А само треугольное число $(10 \times 11) : 2 = 55$.

В общем случае k-е треугольное число равно

$$\frac{k(k+1)}{2}.$$

Задание для самостоятельной работы

1. Разработайте программу для расчета значения k-го треугольного числа по полученной формуле.

Задача 2 “Определить порядковый номер заданного значения треугольного числа *n*”

Очевидное решение такое — последовательно рассчитывать значение 1-го, 2-го, … треугольного числа (запоминая номер) до тех пор, пока не встретится заданное число n. При этом расчеты можно проводить по полученной чуть выше формуле.

Поскольку, как говорят в программирования, “число итераций” (число повторов действий) неизвестно, то необходимо использовать в программе оператор цикла с условием. Возможны два варианта программы:

1) с применением оператора цикла с предусловием (известно условие продолжения работы оператора):

```
алг Номер_треугольного_числа_Вариант1_1
нач цел n, номер
вывод нс, "Задайте треугольное число "
ввод n
номер := 1
нц пока div(номер * (номер + 1), 2) < n
  |Проверяем число со следующим номером
  номер := номер + 1
кц
вывод нс, "Номер этого числа: ", номер
кон
```

— где `div` — функция, возвращающая целую часть частного отделения своего первого аргумента на второй (в других языках программирования для этого используется не функция, а специальная операция);

2) с применением оператора цикла с постусловием (известно условие окончания работы оператора):

```
алг Номер_треугольного_числа_Вариант1_2
нач цел n, номер
вывод нс, "Задайте треугольное число "
ввод n
номер := 0
нц
  номер := номер + 1
кц_при div(номер * (номер + 1), 2) = n
вывод нс, "Номер этого числа: ", номер
кон
```

Обратим внимание на начальное значение величины *номер* (заданное значение n может быть равно 1).

Оказывается, что для решения задачи можно отказаться от использования в программе оператора цикла! Можно рассуждать так. Нам известна зависимость между значением треугольного числа n и его номером *номер*:

$$n = \frac{номер(номер+1)}{2}.$$

Ее можно записать в виде:

$$2n = номер(номер + 1) = номер^2 + номер.$$

Видно, что $2n$ несколько больше, чем квадрат числа *номер*. Поэтому можно сказать, что значение *номер* равно целой части квадратного корня из $2n$.

Соответствующая программа:

```
алг Номер_треугольного_числа_Вариант_2
нач цел n, номер
вывод нс, "Задайте треугольное число "
ввод n
|Рассчитываем квадратный корень
корень := sqrt(2 * n)
|Определяем его целую часть
номер := int(корень)
вывод нс, "Номер этого числа: ", номер
кон
```

Задача 3 “Определить, является ли заданное натуральное число *n* треугольным”

Здесь также можно последовательно рассчитывать значение 1-го, 2-го, …, треугольного числа. Но до каких пор? Это можно делать, пока рассчитанное число меньше заданного n:

```
ммномер := 1
нц пока div(номер * (номер + 1),2) < n
  номер := номер + 1
кц
```

Указанный оператор цикла с предусловием закончит работу, когда рассчитанное число станет больше заданного *n* либо станет равно ему. Значит, после этого возможны два варианта ответа:

```
|Сравниваем
если div(номер * (номер + 1), 2) = n
  то
    вывод нс, "Заданное число является
               треугольным"
  иначе
    вывод нс, "Заданное число
               треугольным не является"
все
```

Задание для самостоятельной работы

2. Разработайте программу для решения задачи 3, в которой используется оператор цикла с постусловием.

Задача 4 “Определить количество треугольных чисел, не превышающих заданное натуральное число *m*”

Задачу можно решить, проверив каждое из чисел от 1 до *m*, является ли оно треугольным (см. предыдущую задачу), и в случае положительного ответа учесть это в искомом количестве. Соответствующая программа:

```
алг Количество_треугольных_чисел
нач цел m, n, номер, кол
вывод нс, "Задайте значение числа m "
ввод m
|Начальное значение искомого количества
кол := 0
|Проверяем каждое из чисел (см. выше)
нц для n от 1 до m
  номер := 1
  нц пока div(номер * (номер + 1), 2) < n
    номер := номер + 1
  кц
  если div(номер * (номер + 1), 2) = n
    то
      |Встретилось очередное
      |треугольное число
       кол := кол + 1
  все
кц
вывод нс, "Количество треугольных
           чисел = ", кол
кон
```

Читаемость (понятность) программы улучшается, если создать вспомогательную функцию логического типа, определяющую, является ли ее аргумент треугольным числом. Такая функция оформляется на основе программы решения задачи 2:

```
алг лог Треугольное(арг цел n)
нач цел номер
номер := 1
нц пока div(номер * (номер + 1), 2) < n
  номер := номер + 1
кц
|После этого можем определить
|значение функции (знач)
если div(номер * (номер + 1), 2) = n
  то
    знач := да
  иначе
    знач := нет
все
кон
```

С использованием созданной функции основная программа оформляется кратко и понятно:

```
алг Количество_треугольных_чисел_Вариант_2
нач цел m, n, кол
вывод нс, "Задайте значение числа m "
ввод m
кол := 0
|Проверяем каждое из чисел,
нц для n от 1 до m
  |используя функцию
  если Треугольное(n)
    то
      кол := кол + 1
  все
кц
вывод нс, "Количество треугольных
           чисел = ", кол
кон
```

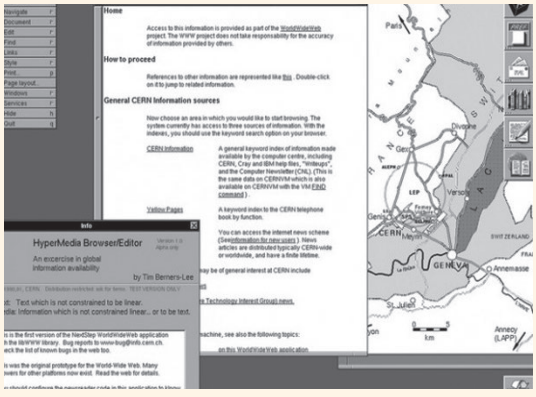
Задание для самостоятельной работы

3. Определите количество трехзначных треугольных чисел.

Оказывается, что для решения задачи 3 можно не использовать в программе ни вложенный цикл (как в варианте 1), ни вспомогательную функцию (как в последнем варианте)! Как именно — установите самостоятельно и разработайте соответствующую программу.

ИСТОРИЯ ИНФОРМАТИКИ

25 декабря 2015 года исполнилось 25 лет со дня запуска первого сайта в Интернете. Ресурс первоначально был закрыт для широкой публики, и его первыми посетителями стали сотрудники Европейского совета по ядерным исследованиям ЦЕРН (от французского названия центра — *Conseil Européen pour la Recherche Nucléaire*) — крупнейшей в мире организации, занимающейся исследованиями в области физики элементарных частиц. Сайт объяснял суть разработанного британским ученым Тимом Бернерсом-Ли гипертекстового проекта World Wide Web (WWW), который позволял обмениваться информацией с коллегами, используя механизм гиперссылок в документах.



Так выглядел первый сайт

Мухоловка считает

Ученые установили, что хищное растение — мухоловка умеет считать.

Ученые из Германии выяснили, что растения, называемые "венерины мухоловки", способны считать количество прикосновений к их ловушке.

Вене́рина мухоловка[2] (лат. Dionaea muscipula) — вид хищных растений семейства Росянковых (Droseraceae). Это небольшое травянистое растение с розеткой из 4–7 листьев, которые растут из короткого подземного стебля. Листья размером от трех до семи сантиметров (в зависимости от времени года). Питается насекомыми, которых ловит в ловушку, образуемую краями длинных листьев. (Посмотреть процесс захлопывания ловушки в виде замедленной видеосъемки можно на сайте *https://ru.wikipedia.org/wiki/Венерина_мухоловка*.)

Райдер Гедрих из университета города Вюрцбург (Германия) и его коллеги раскрыли удивительную способность мухоловки считать и выяснили, что некоторые числа означают для нее, наблюдая за тем, как венерины мухоловки ловили и поедали кузнечиков в лабораторной оранжерее.

Перед экспериментом ученые прикрепляли к поверхности мухоловки несколько электродов, после чего запускали в сосуд с ней кузнечика, который через некоторое время попадал в "объятия" растения. Иногда ученые проверяли хищную представительницу флоры на бдительность, пытаясь заставить ее захлопнуть ловушку после нескольких нажатий на ее поверхность.

Эти опыты раскрыли несколько интересных вещей. К примеру, оказалось, что растение умеет считать и что оно ассоциирует некоторые этапы своего цикла "охоты" и пищеварения с тем, как далеко оно продвинулось в своей "считалочке".

Как рассказывают ученые, мухоловка использует счет до двух для того, чтобы защищать себя от ложных срабатываний — растение никогда не захлопнет ловушку, если ее поверхности не коснуться дважды в течение 20 секунд.

"Умение считать также помогает растению экономить — число касаний информирует мухоловку о том, насколько крупной является ее жертва и как много нутриентов (биологически значимых элементов — белков, минералов и др. — **Прим. ред.**) она содержит. Это помогает растению находить баланс между расходом энергии на охоту и получаемым объемом питательных веществ", — заявил Райнер Гедрих.

Во-вторых, когда насекомое попадает в "объятия" мухоловки, растение не спешит начать переваривать его, пока не убедится, что жертва прочно закреплена в ловушке — мухоловка ждет, пока членистоногое коснется стенок еще пять раз, и только после этого начинает вырабатывать пищеварительные соки.

Пока жертва еще жива, каждое новое "накручивание" счетчика заставляет растение усиливать сжимающую силу и все сильнее сдавливать насекомое. Подобный "пошаговый" подход к его умерщвлению, объясняет Гедрих, позволяет мухоловке экономить силы и определять, как много пищеварительных соков, чье производство крайне дорого с энергетической точки зрения, понадобится для переваривания жертвы.

В настоящее время ученые изучают структуру генома мухоловок, пытаясь найти в нем гены, отвечающие за работу этих встроенных "счётов".

Интересно, что научное название вида (muscipula) переводится с латыни как "мышеловка" и, вероятно, дано по ошибке…

По материалам сайтов *http://www.unian.net/science/1243591-uchenyie-ustanovili-chto-hischnoe-rastenie-muholovka-umeet-schitat.html* и *https://ru.wikipedia.org/*

ЦИФРОВОЙ МИР

Побит рекорд скорости передачи данных по протоколу TCP

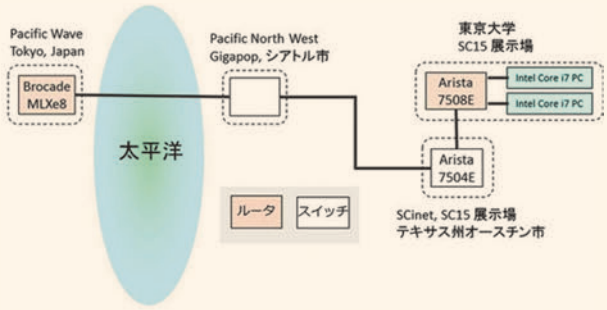
Как известно, одним из основных протоколов, на основе которых работает Интернет, является протокол передачи данных TCP (***T**ransmission **C**ontrol **P**rotocol*). Он разрабатывался в те времена, когда Всемирная сеть еще только проектировалась. Его особенность, в отличие от другого основного протокола, UDP, — гарантированная доставка данных с уведомлением отправителя об их получении. Пока речь шла о скоростях, измеряемых в килобитах, а затем в мегабитах и десятках мегабит, все шло неплохо. Даже стандарты 100 и 1000 Мбит/с были освоены сравнительно легко.

Но информационная сеть, покрывшая уже практически всю планету, растет. Растет вместе с ней и пропускная способность основных каналов. И на скоростях порядка десятков Гбит/с использование TCP становится задачей весьма нетривиальной — протокол, разработанный в 1974 году, просто не рассчитан на такие потоки данных. Сейчас вопрос стоит особенно остро, поскольку идет процесс замены 10-гигабитных

[2] Русское название вид получил в честь Венеры — римской богини любви и растений. Английское название вида (Venus's-flytrap, или Venus flytrap, или Venus' flytrap) соответствует русскому.

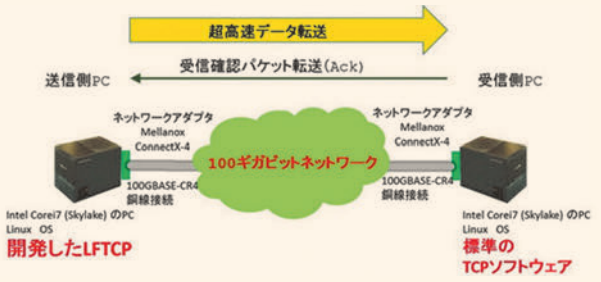
сетевых каналов 100-гигабитными, а предыдущий рекорд передачи с использованием TCP/IP составлял всего 29 Гбит/с: не слишком эффективное использование доступной пропускной способности, даже с учетом избыточности кодирования — менее 50%.

Маршрут рекордной передачи



Экспериментальное оборудование и схема маршрута передачи данных

Но группе исследователей Токийского университета удалось побить этот рекорд, установив устойчивую передачу данных с использованием TCP на скорости 73 Гбита/с, что уже близко к пределу 100-гигабитного сетевого канала. Исследование было проведено летом 2015 года при содействии США при использовании 100-гигабитной сети TransPAC/Pacific Wave. Протокол TCP подвергся усовершенствованию, которое разработчики называют LFTCP (***L**ong **F**at pipe **TCP***). Эта надстройка над TCP и его программным обеспечением позволяет почти полностью задействовать потенциал 100-гигабитных сетей без изменения самой сути протокола. Кроме того, LFTCP имеет открытый программный код, доступный всем исследователям, интересующимся данной темой.



Конфигурация аппаратного и программного обеспечения

Эксперимент, проведенный исследователями из Японии и США, состоялся 16–17 ноября 2015 года. Системы были установлены в Токио и в г. Остин, штат Техас. Их конфигурация была самой обычной, на базе процессора Core i7-6770K, а в качестве сетевого адаптера использовалась карта Mellanox ConnectX-4. Системы работали под управлением CentOS Linux 7.1, для измерения производительности использовался пакет iperf3. Передающая система использовала LFTCP, принимающая — обычный TCP. Задержка прохождения сигнала туда и обратно составила 296 миллисекунд, но это не слишком удивительно: расстояние между Остином и Токио составляет 21 153 километра.

Компьютеры, участвовавшие в проекте (слева), и экспериментальный маршрутизатор

Результаты эксперимента представляют большую ценность, поскольку позволяют понять, по какому пути следует развиваться Всемирной сети для передачи все большего количества данных и ускорения доступа к WWW. Кроме того, один из проектов Токийского университета в области биологии способен генерировать поток данных свыше 50 Гбит/с, так что технология LFTCP очень пригодится ученым всего мира, имеющим дело с огромными массивами данных и нуждающимся в быстрой их передаче от одной исследовательской группы к другой. Ведь в век глобализации научные группы могут работать над одной проблемой, будучи разбросанными по всему миру.

Источник: 3dnews.ru

Уважаемые коллеги!

Для поощрения самых активных участников конкурсов, проводимых в разделе “В мир информатики”, редакция может направить вам электронный вариант диплома.

Заявку на диплом просьба прислать в редакцию электронной (адрес: *vmi@1september.ru*) или обычной почтой в апреле–мае. Оформление дипломов будет проводиться в учебном заведении.

МЫСЛИ

Воспитание — дело трудное, и улучшение его условий — одна из священных обязанностей каждого человека, ибо нет ничего более важного, как образование самого себя и своих ближних.

Сократ, древнегреческий философ

НОВЫЙ ПРОЕКТ

«Первого сентября»



СПЕЦИАЛИСТЫ-ПРАКТИКИ

СВИДЕТЕЛЬСТВО УЧАСТНИКА

О ВОСПИТАНИИ

ОБ ОТНОШЕНИЯХ

О РОДИТЕЛЬСКОЙ ПОЗИЦИИ

О САМООЦЕНКЕ

АБОНЕМЕНТЫ

ОБ ОПТИМИЗМЕ

УДОБНОЕ

О ЦЕЛЯХ

О ЧУВСТВЕ ВИНЫ

О РАБОТЕ

О МИРОВОСПРИЯТИИ

О КОММУНИКАЦИИ

ВРЕМЯ

О ДЕТЯХ

ВЕБИНАРЫ

О ДЕТЯХ С ОВЗ

ВОСТРЕБОВАННЫЕ

О КОНФЛИКТАХ

О СЕМЬЕ

ВИДЕОЗАПИСИ

ТЕМЫ

ДОСТУПНАЯ СТОИМОСТЬ

О МЕТОДАХ ОБУЧЕНИЯ

О ВЫГОРАНИИ

О КАРЬЕРЕ

О ВЗАИМОПОНИМАНИИ

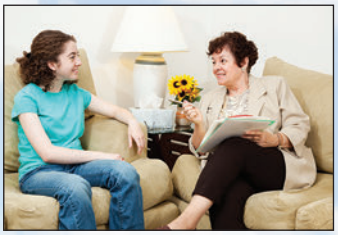
О СТРЕССЕ

ВОПРОСЫ И ОТВЕТЫ ОНЛАЙН

О ЦЕННОСТЯХ

О ЛИЧНЫХ КРИЗИСАХ





Видеозаписи вебинаров на сайте

webinar.1september.ru

«Новые средства – только добавление возможностей, а не замена дидактики и имеющегося опыта»

Электронные формы учебников (ЭФУ) используются в школах страны относительно недавно. Сейчас – период приобретения, накапливания личного опыта в работе с ЭФУ. Важны общие усилия в этом направлении всего педагогического сообщества. Необходимо пробовать, анализировать и обобщать полученный опыт, обращая особое внимание на трудности в использовании ЭФУ – как технические, так и дидактические.

Сегодня уже трудно представить школу без информационно-образовательной среды: без компьютеров, ноутбуков, планшетов, интерактивных досок, проекторов. В этих новых условиях перед учителем стоит задача найти баланс в триаде «устройство – содержание – методика». При этом нельзя пренебрегать и вопросами здоровьесбережения.

Исследования использования ИТ в классе показывают, что учащиеся, пользующиеся компьютером умеренно (до двух часов в день), учатся лучше своих сверстников, использующих компьютер часто (до шести часов в день). У ребят, сидящих перед экраном свыше шести часов в день, могут даже возникнуть психологические проблемы.

Следовательно, оправдан умеренный подход в использовании технологий.

Эффективное использование ИКТ-среды позволяет лучше вовлекать ученика в работу с учетом его образовательных возможностей и потребностей для достижения новых образовательных результатов.

Электронный учебник, развиваясь, все больше выходит в центр образовательной ИКТ-среды. Методические требования к ЭФУ:

- контент учебника должен быть избыточным и вариативным (чтобы отвечать нуждам и потребностям как слабого, так и сильного ученика);
- учебник должен быть интерактивным и мультимедийным;
- учебник должен содержать разнообразные виды КИМов и домашних заданий;
- учебник должен позволять интеграцию или встраивание других материалов, в т. ч. из открытых источников, собственных.

Существующие ЭФУ вполне покрывают спектр пожеланий и требований учителей. Главное – найти верный ход и использовать то, что в ЭФУ уже заложено, чтобы новые формы, новые технологии служили подспорьем мастерству и творчеству педагога.

В электронные формы учебников встроены:

- словарь терминов,
- заметки,
- тренажеры,
- «плитки» – мультимедийные наглядные объекты.

Рассмотрим возможности их использования на уроке.

Словарь терминов – это не новшество ЭФУ. Словари есть и в печатных учебниках. Их можно открыть, термин можно найти, прочитать, выучить, запомнить, выписать в тетрадь… И все-таки электронный учебник позволяет интенсифицировать процесс работы со словарем, поскольку термин и его дефиницию не обязательно переписывать в тетрадь, а можно просто скопировать и поместить в *заметки*. Каждая заметка привязывается к соответствующему параграфу, но есть возможность просмотреть все заметки сразу. Этот удобный инструмент реализован в каждом электронном учебнике всех издательств. Кроме того, в заметки можно вносить свои комментарии, записывать формулы, термины, тезисы, домашние

задания и обращаться к ним в случае необходимости. С помощью заметок ученик создает **свой** конспект учебника в соответствии со **своим** собственным пониманием. Для ученика это возможность осознанного учения, отбора важной информации, ее классификации. Заметки – инструмент, позволяющий подстроить содержание учебника под себя, индивидуализировать учебный процесс.

Электронный учебник ценен наличием *тренировочных тестов* (в профессиональном сообществе их уже называют «*тренажерами*»). За выполнение этих тестовых заданий отметка, как правило, не выставляется. Но ученик сразу видит результат тестирования, видит свои сильные и слабые стороны, понимает, что нужно выучить или повторить. Тренажер помогает выявлять дефициты в знаниях учащегося. Выполняя тесты, фрагменты тестов или отдельные задания, ученик «тренируется», лучше запоминает материал. Можно какие-то вещи, факты запомнить или выучить, обращаясь только к тренажеру, идя к правильному ответу через ошибки, задумываясь, почему тот или иной вариант ответа неверный.

Следует отметить, что в ЭФУ предлагаются разные виды закрытых и условно открытых тестовых заданий. Это и задания с множественным выбором правильного ответа, и задания на выстраивание последовательности, на поиск соответствий. В других заданиях ответ может быть введен в соответствующее поле с клавиатуры.

В электронном учебнике после каждого параграфа имеется некоторое количество проверочных или контрольных вопросов. Из них для каждого ученика формируется свой набор из трех-пяти вопросов, ответ на которые позволяет понять уровень усвоения материала. В конце урока ученики уже могут не подвергаться заключительному фронтальному опросу. Каждый из них просто выполняет свои задания и тут же получает результат.

Важно отметить и эмоциональную составляющую такого быстрого тестирования. Ученик получает мгновенную обратную связь, получает подтверждение или опровержение своих представлений об успешности своего обучения на данном уроке (несколько модифицируется этап рефлексии). Кроме того, неуспевающий ученик имеет возможность не выставлять свое незнание на всеобщее обозрение, как это могло бы быть при заключительном фронтальном опросе. Еще один положительный момент подобного тестирования: в отличие от фронтального опроса, когда возможно охватить лишь некоторое число учащихся, в тестировании с помощью тренажера принимают участие все ученики. Стопроцентный охват с обратной связью за пару минут!

Как отмечалось выше, тренажер можно использовать не только как инструмент (пусть даже и безотметочного) тестирования, но и для тренировки, например, когда нужно довести до автоматизма какие-то навыки, решить некоторое количество похожих, однотипных задач или примеров. Использование технологий позволяют подобные, по своей сути скучные, задания превратить в захватывающее, увлекательное действие. Использование технологий делает возможным создание на уроке так называемой win-win-situation, ситуации без проигравших. Отметки за выполнение заданий не выставляются, но оценивание, фиксация верных и неверных решений происходит. К примеру, на планшетах учеников, правильно выполнивших задание, может появиться сообщение «Молодец!» (как это было реализовано на уроке алгебры в 7-м классе гимназии № 1520 им. Капцовых учителем А.В. Дорониным. См.: http://www.youtube.com/watch?v=NOFhYThwEyM). Учащиеся, не получившие такого сообщения или получившие сообщение «Есть ошибки», понимают, что что-то выполнили не так, сами ищут ошибку или повторно выполняют задание. В подобной ситуации ученики учатся оценивать себя, находить и исправлять свои ошибки. Ученик анализирует свои шаги, действия – проводит рефлексию.

Мгновенная обратная связь, которая возможна при использовании ЭФУ, – это один из ключевых моментов в сегодняшнем обучении. Она позволяет этап рефлексии на уроке сделать более осознанным, осмысленным. Ученик видит, с чем он справился или какой учебной цели он пока не достиг, и пытается найти путь для ее достижения.

ЭФУ дает большие возможности для работы с иллюстративным материалом. *Наглядность* важна при работе как с юными обучающимися, так и со старшеклассниками. Мультимедийность ЭФУ и любые другие технологические возможности электронных учебников – не самоцель. Все имеет смысл в ракурсе достижения цели урока и методических задач, решаемых учениками самостоятельно.

В заключение хотелось бы пригласить коллег-учителей попробовать приобрести электронный учебник, открыть его и для начала не искать, чем он отличается от того, чего вы от него ждали, а найти в нем то, что могло бы помочь в проведении умных, интересных, захватывающих уроков.

Bring Your Own Device

ЭЛЕКТРОННЫЙ УЧЕБНИК – ЭТО СОДЕРЖАНИЕ ПЛЮС УСТРОЙСТВО

Для воспроизведения электронных учебников требуется устройство (планшет, ноутбук, лэптоп, стационарный компьютер и т.п.).

Современные электронные формы учебников проигрываются на широком спектре устройств и практически на всех популярных операционных платформах. Таково требование Министерства образования и науки РФ, выполнение которого необходимо для внесения электронного учебника в Федеральный перечень.

И если обеспечить школу электронными учебниками (лицензиями на пользование) в любом необходимом объеме (любой тираж) не составляет проблем, то снабдить всех учеников класса мобильными устройствами, соответствующими нормам СанПиНа и техническим спецификациям программного комплекса, – непросто.

Это проблема не только нашей страны, это всеобщая проблема, пришедшая к нам вместе с новыми технологиями. Где-то эту проблему пытаются решить через централизованные закупки необходимых устройств хотя бы в пилотные школы, но нарастающий всемирный тренд решения этой проблемы – технологического обеспечения работы с электронными учебниками – получил говорящее название «Bring Your Own Device», что в переводе на русский язык означает: «принеси в школу, в офис своё собственное устройство».

Гипотеза состоит в том, что универсальные мобильные устройства во всё в большем количестве приобретаются пользователями для собственных нужд: личный органайзер, игры, чтение книг, электронная почта, интернет, соцсети.

Практически все устройства, входящие в класс «планшет», достаточно функциональны не только для обслуживания личных (персональных) задач пользователя в повседневной жизни, но и для учебных целей, включая воспроизведение электронных учебников или обслуживание электронной образовательной среды. Исследования рынка подтверждают, что у всё большего количества старшеклассников либо уже есть планшеты, ноутбуки, лэптопы, либо родители готовы их приобрести, особенно для нужд учёбы.

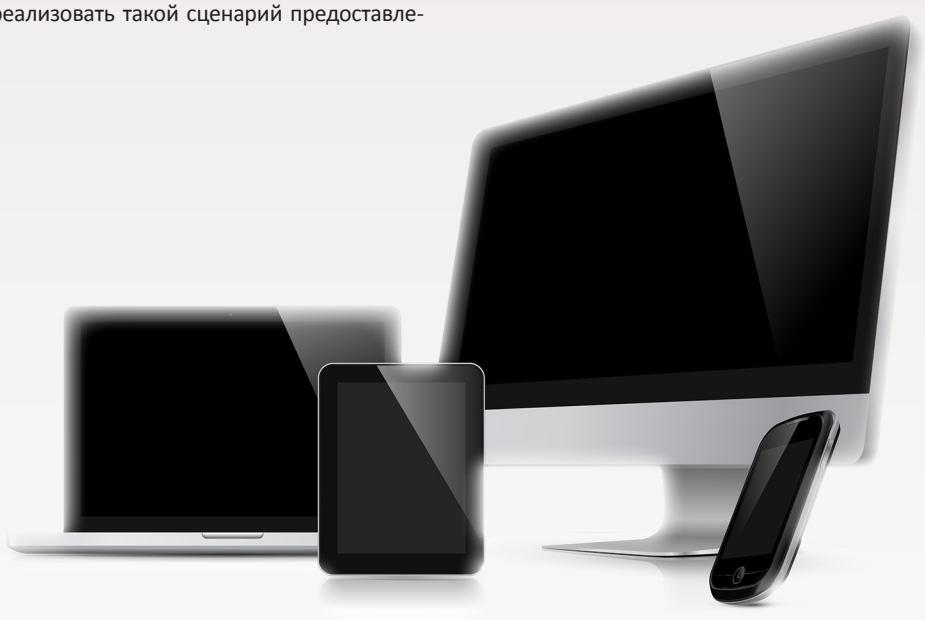
Так или иначе, современный сценарий централизованного предоставления электронных учебников ученикам в классе может выглядеть следующим образом: школа закупает лицензии на использование электронных учебников, а учитель в классе объявляет, что теперь есть возможность передать ЭФУ на устройства тем ученикам, кто готов использовать личные устройства в учебных целях.

В рамках проекта Министерства образования Московской области «Оказание услуги по обеспечению доступа обучающихся общеобразовательных организаций Московской области к электронным учебникам и электронным приложениям к учебникам» был создан сайт-витрина электронных учебников, который поддерживает такой сценарий передачи электронных учебников ученикам. То есть задача витрины ibook.1september.ru – предоставить электронные учебники для использования не только на школьных устройствах, но и на личных устройствах учителей и учеников. Пошаговая инструкция на сайте – текстовая и видео – позволяет каждому учителю Московской области в течение всего 2016 года реализовать такой сценарий предоставления электронных учебников ученикам своего класса за счет бюджета области.

Успешность подхода «bring our own device» зависит от наличия личных планшетов у учеников и от их готовности использовать эти устройства не только в игровых, но и в учебных целях. Проверка актуальности такого подхода и составляет одну из целей проекта по обеспечению учеников Московской области электронными учебниками (электронными формами учебников) в 2016 году. Проект инициирован Министерством образования Московской области и реализуется в сотрудничестве с Издательским домом «Первое сентября».

Какое количество устройств на руках учеников в классе является достаточным для успешного урока? Это предмет исследования и опыта. В следующих статьях мы рассмотрим сценарии уроков, когда устройства с электронной формой учебника есть:

1) только у учителя,
2) у малого количества учащихся,
3) у половины учеников,
4) у всех учеников,
5) когда устройства есть у учителя и всех учеников, и эти устройства связаны в единую технологическую сеть.



Более подробная информация и страница для обмена опытом на сайте digital.1september.ru/etutorial