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

ИНФОРМАТИКА

издательский дом 1september.ru
Первое сентября
ноябрь 2011
ИНФОРМАТИКА Подписка: «Роспечать» — 32291 (бумажная версия), 19179 (электронная); «Почта России» — 79066 (бумажная версия), 12684 (электронная)

О НОМЕРЕ

Этот номер, возможно, получился несколько суховатым и пресным, зато, надеемся, полезным и своевременным. До ЕГЭ и ГИА еще достаточно времени, и если вы преподаете в этом году в выпускных классах, “сухие” и “пресные” материалы вам скорее всего таковыми не покажутся. Отметим также, что большинство типовых задач, используемых для подготовки к решению задач С4, это не “задачи для С4”, а задачи “сами по себе” — базовые алгоритмические задачи, которыми можно заниматься на обычных уроках, в обычной учебной деятельности, а не только исключительно с целью подготовки к экзаменам.

Также хотим обратить внимание на вкладку “В мир информатики” — в ней много интересного. В частности — две очень содержательные и увлекательные “исторические” заметки.

В НОМЕРЕ

НА ДИСКЕ

ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ:

- Исходные коды программ
- Презентации к статьям номера
- Электронная версия номера и подарок — подшивка “Информатики” за 2006 год

Уважаемые подписчики бумажной версии журнала “Информатика”!

В течение второго полугодия 2011 г. вы можете получать и электронную версию нашего журнала в Личном кабинете. Электронная версия в Личный кабинет всегда доставляется точно в срок — 1-го числа каждого месяца.

1. Зайдите на интернет-сайт *www.1september.ru*.
2. Зарегистрируйте Личный кабинет (если у вас его еще нет).
3. В Личном кабинете в разделе “Издания/Коды доступа” введите код SE-00332-68580.

Номера будут доступны в разделе “Издания/Получение”.

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ: по каталогу “Роспечати”: 32291 (бумажная версия), 19179 (электронная версия); “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 3000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Главный редактор: Артем Соловейчик (генеральный директор)
Коммерческая деятельность: Константин Шмарковский (финансовый директор)
Развитие, IT и координация проектов: Сергей Островский (исполнительный директор)
Реклама и продвижение: Марк Сартан
Мультимедиа, конференции и техническое обеспечение: Павел Кузнецов
Производство: Станислав Савельев
Административно-хозяйственное обеспечение: Андрей Ушков
Главный художник: Иван Лукьянов
Педагогический университет: Валерия Арсланьян (ректор)

ГАЗЕТА ИЗДАТЕЛЬСКОГО ДОМА
Первое сентября – Е.Бирюкова

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА
Английский язык – А.Громушкина
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – О.Коротова
Дошкольное образование – М.Аромштам
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – М.Сартан
История – А.Савельев
Классное руководство и воспитание школьников – О.Леонтьева
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Управление школой – Я.Сартан
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школьный психолог – И.Вачков

УЧРЕДИТЕЛЬ: ООО “ЧИСТЫЕ ПРУДЫ”

Зарегистрировано ПИ № ФС77-44341 от 22.03.2011 в Министерстве РФ по делам печати
Подписано в печать: по графику 10.10.2011, фактически 10.10.2011
Заказ №
Отпечатано в ОАО “Чеховский полиграфический комбинат” ул. Полиграфистов, д. 1, Московская область, г. Чехов, 142300

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

Месяц потерь

В хмуром и пасмурном октябре не стало двух выдающихся специалистов-компьютерщиков. Оба они оказали огромное влияние на то, как выглядит сегодня компьютерный мир.

В позднее советское время у многих из нас сформировалось стойкое отвращение к штампам. Даже таким трагическим по сути, как “страна понесла невосполнимую утрату…”. Мы почти разучились чувствовать истинное человеческое горе, которое стояло за обезличенными словами дикторов программы “Время”.

Но, провожая хмурый и пасмурный месяц октябрь, я вдруг понял, что именно стандартная штампованная фраза во всей ее истинной трагичности и приходит мне в голову: “мировая информатика понесла невосполнимые утраты…”.

Стив Джобс (слева), 2007 г.

Деннис Ритчи (в центре), 1999 г.

Уверен, что даже если бы в начале этой заметки не было фотографий, дочитав до этих слов, одну фамилию смог бы назвать каждый. Кончина Стива Джобса, пусть, к сожалению, и давно ожидаемая и сообществом, и рынком, стала истинным горем не только для поклонников продукции Apple. Второго человека многие, возможно, не узнали. И не узнали, даже прочитав подпись под фотографией. Уход из жизни Денниса Ритчи остался практически незамеченным профессиональным сообществом. Но убежден, сам Стив Джобс ценил роль Ритчи в информатике ничуть не меньше своей собственной. Оба они оказали огромное влияние на то, как выглядит сегодня компьютерный мир.

В 1983 г. империя Стива Джобса — самая дорогая сегодня компьютерная компания мира — не только не была империей, но и само будущее Apple было под большим вопросом. До триумфа было очень далеко. В этом же году Деннис Ритчи уже был назван лауреатом премии Тьюринга — самой престижной награды, которой удостаиваются компьютерные специалисты. Эту премию Ритчи получил “за разработку общей теории операционных систем и, в частности, за создание UNIX”. В этой формулировке — несколько академичной, как и положено высокой научной награде, — не было упоминания о любимом творении Ритчи, которое сделало возможным создание Unix. Речь идет о языке программирования Си.

Далеко не все программисты даже среднего и старшего поколений помнят Ритчи как соавтора Unix, но нет среди профессионалов ни одного, кто бы не штудировал в свое время “Библию языка Си” — базовый учебник, написанный Ритчи в соавторстве с Брайаном Керниганом и потому часто называемый K&R.

В хмуром и пасмурном октябре не стало двух выдающихся специалистов-компьютерщиков. Мир потерял.

Фотографии с сайта ***wikimedia.org***

Что надо уметь для решения задач С4 из ЕГЭ по информатике

Д.М. Златопольский,
Москва

Анализ задач С4 из демонстрационных вариантов ЕГЭ по информатике и ИКТ прошлых лет [1–4], а также вариантов заданий от разработчиков контрольно-измерительных материалов для ЕГЭ [5–6] показывает, что для их решения необходимо знать методику решения ряда типовых вспомогательных задач. Такая методика описывается в данной статье. При анализе используется школьный алгоритмический язык. Русский синтаксис этого языка и большое число комментариев делает программы максимально понятными и легко переносимыми на любой другой язык программирования. В ряде случаев приводятся также соответствующие фрагменты программ решения задач на языке Паскаль. Заметим попутно, что ряд рассмотренных задач встречаются в ЕГЭ так же, как задачи С2.

Прежде чем рассматривать типовые задачи и методику их решения, заметим следующее. В большинстве задач С4 в ЕГЭ их авторы предлагают написать “эффективную, в том числе и по используемой памяти, программу” или “как можно более эффективную программу”. При этом некоторые критерии эффективности приводятся только в демонстрационных вариантах экзаменов и в решении ряда задач в [5].

Возникает ряд вопросов, связанных с эффективностью программ.

1. Влияет ли на эффективность программы начальное присваивание переменным (в том числе элементам массивов) нулевого или “пустого” значения в системах программирования, в которых такое присваивание не является обязательным? Ответ — да, влияет (в худшую сторону — каждый новый оператор увеличивает длину кода программы). В программах, приводимых в [1–6], такое присваивание проводится. Это же будем делать и мы (учитывая, что в программировании это является “правилом хорошего тона”).

2. Является ли программа, в которой перед вводом значений переменных на экран выводятся сообщения, облегчающие ввод, менее эффектив-

ной, чем программа, в которой такие сообщения не выводятся? Учитывая, что в программах, представленных в [1–6], сообщения не выводятся, мы также этого делать не будем.

Что касается вывода сообщений в ответах, то в [5] такие сообщения выводятся, в [6] — нет. При решении задач из соответствующих книг мы будем поступать аналогично их авторам.

3. Влияет ли на эффективность программы на языке Паскаль использование для числовой переменной типа `integer` вместо типа `byte`, хотя возможные значения этой величины соответствуют последнему типу? В программах, приводимых в [1–6], возможное уменьшение объема памяти, выделяемого для хранения значений переменных величин, почему-то не проводится.

4. Является ли программа, в которой используются "длинные" имена переменных, менее эффективной, чем программа, в которой имена максимально сокращены? Ответ — нет. Поэтому мы, в отличие от программ в [1–6], будем использовать имена без максимальных сокращений, что делает смысл соответствующих величин более понятным.

5. Является ли программа, в которой в целях экономии памяти одно и то же имя переменной используется для величин с разным смыслом, более эффективной, чем программа, где для таких величин используются два разных имени, что делает программу более понятной? Ответ — да, является. В ряде задач авторы задач ЕГЭ используют имя *s* для обозначения как суммы значений, так и среднего значения (которое рассчитывается как `s := s/k`). Поэтому подобное будем делать и мы, хотя не считаем это логичным с точки зрения понимания программы.

6. Программа, в которой выражения для расчета значений ответов указываются в операторе вывода, безусловно, является более эффективной с точки зрения используемой памяти, чем программа с предварительным расчетом значения ответа в отдельной переменной. Так, с выражением для расчета значений вывод ответов оформлен в решении ряда задач в [5]. И хотя мы считаем это методически неправильным (программа является "аналогом" алгоритма, в котором расчет значений величин и их вывод — разные этапы), при разборе задач из указанной книги мы будем оформлять ответы так же.

1. Группа задач на выделение частей строки

В задачах 1.1–1.4 обрабатывается задаваемая во время выполнения программы строка символов с именем *строка*, имеющая в общем случае следующую структуру:

С	и	…	л		Ц	в	е	…	Я					
1-е слово				*П*	*2-е слово*					*П*	*k-е слово*			

где *П* — пробел.

1.1. Выделение первого слова

Соответствующий фрагмент имеет вид:

```
|Формируемое слово (начальное значение)
слово1 := ""
|Номер очередного символа строки
номер_симв := 1
|Пока не встретится 1-й пробел
нц пока строка[номер_симв] <> " "
  |Добавляем текущий символ к
  |"старому" значению величины слово1
  слово1 := слово1 + строка[номер_симв]
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
кц
```

Аналогичный фрагмент **на языке Паскаль**:

```
slovo1 := '';
{Формируемое слово (начальное значение)}
nomer_simv := 1;
{Номер очередного символа строки}
while stroka[nomer_simv] <> ' ' do
{Пока не встретится 1-й пробел}
begin
  {Добавляем текущий символ к "старому"
   значению величины слово1}
   slovo1 := slovo1 + stroka[nomer_simv];
   {Переходим к следующему символу}
   nomer_simv := nomer_simv + 1
end;
```

Можно также применить оператор цикла с постусловием:

```
слово1 := ""
номер_симв := 1
нц
  слово1 := слово1 + строка[номер_симв]
  номер_симв := номер_симв + 1
кц_при строка[номер_симв] = " "
|Встретится 1-й пробел
```

Язык Паскаль

```
slovo1 := '';
nomer_simv := 1;
repeat
  slovo1 := slovo1 + stroka[nomer_simv];
  nomer_simv := nomer_simv + 1
until stroka[nomer_simv] = ' '
{Встретится 1-й пробел}
```

Заметим здесь же, что при оформлении фрагмента в виде:

```
слово1 := ""
номер_симв := 0
|Начальный номер равен нулю
нц
  номер_симв := номер_симв + 1
  слово1 := слово1 + строка[номер_симв]
кц_при строка[номер_симв] = " "
```

— пробел будет включен в состав первого слова.

Если в языке программирования имеется функция, возвращающая позицию первого вхождения в строку той или иной подстроки (а в языке Паскаль такие функции есть), то для решения задачи можно не использовать оператор цикла:

```
slovo1 := Copy(stroka, 1,
                POS(stroka, ' ') - 1);
```

Также можно обойтись без использования оператора цикла, если длина первого слова известна (и равна *длина1*/*dlina1*):

— в программе на школьном алгоритмическом языке:

```
слово1 := строка[1 : длина1]
```

— в программе на Паскале:

```
slovo1 := Copy(stroka, 1, dlina1);
```

Программа на языке Паскаль может быть также улучшена за счет того, что вводимые символы строки можно считывать сразу во время их ввода, используя оператор `read` с параметром символьного типа:

```
repeat
 {Считываем очередной введенный символ}
 read(c);
 {и добавляем его к "старому" значению
  величины slovo1}
 slovo1 := slovo1 + c
until c = ' ';
```

Это позволяет отказаться от использования переменной *nomer_simv* и от переменной *stroka* строкового типа. Однако в этом случае пробел будет включен в состав первого слова.

В ряде задач такой способ обработки последовательности символов позволяет отказаться также от использования переменной строкового типа. Обратим внимание на то, что в решениях, приведенных в [1–6], используется именно этот способ — он более эффективен с точки зрения использования памяти по сравнению с предварительным запоминанием всей строки в целом.

Аналогичный способ можно применить и в программах на школьном алгоритмическом языке, в случае обработки последовательности символов латинского алфавита (а именно это имеет место в некоторых задачах из [5–6]). Соответствующий фрагмент программы имеет вид:

```
слово1 := ""
нц
 |Считываем очередной введенный символ
 симв := символ(клав)
 |и добавляем его к "старому" значению
 |величины слово1
 слово1 := слово1 + симв
кц_при симв = " "
```

— где *симв* — величина символьного (в терминах школьного алгоритмического языка) типа, *клав* — функция, возвращающая код нажатой клавиши, *символ* — функция, возвращающая символ с кодом, равным ее параметру.

1.2. Выделение второго слова

Для этого следует сначала пропустить первое слово:

```
номер_симв := 1 |Номер очередного символа
строки
нц
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
  |пока не встретится 1-й пробел
кц_при строка[номер_симв] = " "
|Величина номер_симв указывает на 1-й пробел
```

После этого можно сформировать второе слово:

```
|Формируемое слово (начальное значение)
слово2 := ""
|Переходим к 1-му символу 2-го слова
номер_симв := номер_симв + 1
нц пока строка[номер_симв] <> " "
|Пока не встретится 2-й пробел
|Добавляем текущий символ к "старому"
|значению величины слово2
  слово2 := слово2 + строка[номер_симв]
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
кц
```

В программе на языке Паскаль можно также первое слово пропустить, использовав функцию Pos (см. выше), — установить значение величины *nomer_simv*, равное номеру символа после первого пробела.

Кроме того, в программе на этом языке, как и применительно к предыдущей задаче, вводимые символы строки можно считывать сразу во время их ввода, используя оператор `read` с параметром символьного типа:

```
{Пропускаем первое слово}
repeat
 read(c)
until c = ' ';
{Формируем второе слово}
repeat
 {Считываем очередной введенный символ}
 read(c);
 {и добавляем его к "старому" значению
  величины slovo1}
 slovo1 := slovo1 + c
until c = ' ';
```

Обращаем внимание на тот факт, что и здесь второй пробел будет включен в состав второго слова.

1.3. Выделение двух первых слов как единой величины

В программах на Паскале, приведенных в [1–6] для вариантов, в которых такая задача возникает, пробел между словами и пробел после второго слова включается в выделяемую величину. Поступая аналогично, можем использовать отмеченный при решении задачи 1.1 вариант:

```
слово12 := ""
номер_симв := 0
|Начальный номер равен нулю
нц
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
  |Переходим к следующему символу
  слово12 := слово12 + строка[номер_симв]
кц_при строка[номер_симв] = " "
|Первое слово и пробел после него
|включены в состав величины слово
|Обрабатываем второе слово
нц
  номер_симв := номер_симв + 1
  слово12 := слово12 + строка[номер_симв]
кц_при строка[номер_симв] = " "
|Второй пробел также учтен
```

— где *слово*12 — формируемая величина.

Можно также несколько сократить фрагмент, использовав вложенный оператор цикла:

```
слово12 := ""
номер_симв := 0
нц для i от 1 до 2
 нц
  номер_симв := номер_симв + 1
  слово12 := слово12 + строка[номер_симв]
 кц_при строка[номер_симв] = " "
кц
```

Конечно, когда выделяемых слов три и более, применение вложенного оператора цикла дает больший эффект.

Язык Паскаль

```
{Формируем первое слово}
repeat
 {Считываем очередной введенный символ}
 read(c);
 {и добавляем его к "старому" значению
  величины slovo12}
 slovo12 := slovo12 + c
until c = ' ';
{Добавляем второе слово}
repeat
 read(c);
 slovo12 := slovo12 + c
until c = ' ';
{Второй пробел также учтен}
```

1.4. Выделение последнего слова

В программе на школьном алгоритмическом языке задача решается аналогично задаче 1.2:

1) пропускаются все слова, кроме последнего;

2) формируется последнее слово.

В программе на языке Паскаль после пропуска всех слов, кроме последнего, его можно считать, применив оператор `readln` с параметром, соответствующим искомому слову: `readln(slovo);`

1.5. Выделение числа после первого слова

Будем считать, что искомым числом строка заканчивается.

Для решения задачи необходимо сначала пропустить первое слово и пробел после него:

```
|Пропускаем 1-е слово
номер_симв := 1
|Номер очередного символа строки
нц
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
  |пока не встретится 1-й пробел
кц_при строка[номер_симв] = " "
|Величина номер_симв указывает
|на 1-й пробел
|Пропускаем его
номер_симв := номер_симв + 1
```

После этого надо получить последовательность символов-цифр и преобразовать ее в число:

```
|Получаем символы строки, представляющие
|собой запись числа,
|путем копирования соответствующей
|подстроки и преобразуем ее в число
число := лит_в_цел(строка[номер_симв :
                   длин(строка)], успех)
```

— где `длин` — функция, возвращающая общее количество символов в строке, — функция `лит_в_цел`, возвращающая числовое представление числа — ее аргумента, представленного символами-цифрами (подробнее об этой функции см. раздел 5).

Программа на языке Паскаль существенно упрощается, если после пропуска первого слова и пробела считать оставшиеся символы-цифры как единое число, записав оператор `readln`:

```
readln(chislo);
```

Считанное значение присваивается величине числового типа, т.е. преобразование строки в число не требуется.

Если длина числа известна (и равна *длина*1/*длина*1), то искомое значение можно получить без использования оператора цикла, учитывая, что “местоположение” числа в строке известно:

```
число := лит_в_цел(строка[длин(строка) -
       длина1 + 1 : длин(строка)], успех)
```

В задачах 1.5–1.8 обрабатываемая строка символов с именем *строка* имеет в общем случае следующую структуру:

С	и	…	л		Ц	в	е	…	Я																							
1-е слово				*П*	*2-е слово*					*П*	*k-е слово*					*П*	*1-е число*				*П*	*2-е число*					*П*	*3-е число*				

где *П* — пробел, *1-е число*, *2-е число* и *3-е число* — символы строки, представляющие собой запись чисел.

Соответствующий оператор в программе на языке Паскаль:

```
Val(Copy(stroka, Length(stroka) –
    dlina1 + 1, dlina1), chislo, code);
```

1.6. Выделение числа после второго слова

Будем считать, что искомым числом строка заканчивается.

Задача решается аналогично предыдущей, с той разницей, что пропустить необходимо два первых слова и пробел после второго слова:

```
|Пропускаем 1-е слово
номер_симв := 1
|Номер очередного символа строки
нц
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
  |пока не встретится 1-й пробел
кц_при строка[номер_симв] = " "
|Величина номер_симв указывает
|на 1-й пробел
|Пропускаем 2-е слово
нц
  |Переходим к следующему символу
  номер_симв := номер_симв + 1
  |пока не встретится 1-й пробел
кц_при строка[номер_симв] = " "
|Пропускаем 2-й пробел
номер_симв := номер_симв + 1
|Получаем символы строки, представляющие
|собой запись числа,
|путем копирования соответствующей
|подстроки и преобразуем ее в число
число := лит_в_цел(строка[номер_симв :
                   длин(строка)], успех)
```

В программе на языке Паскаль можно также после пропуска слов и пробелов считать оставшиеся символы-цифры как единое число:

```
readln(chislo);
```

Если длина числа известна, то искомое числовое значение можно получить без использования оператора цикла (см. предыдущую задачу).

1.7. Выделение двух чисел после второго слова

Будем считать, что количество цифр в каждом числе известно и равно, соответственно, *длина*1(*dlina*1) и *длина*2(*dlina*2).

Соответствующий фрагмент:

```
|Пропускаем два первых слова
|и второй пробел
… (см. предыдущую задачу)
|Получаем первое число
число1 := лит_в_цел(строка[номер_симв :
          номер_симв + длина1 - 1], успех)
|Пропускаем первое число и пробел
|после него
номер_симв := номер_симв + длина1 + 1
|Получаем второе число
число2 := лит_в_цел(строка[номер_симв :
          номер_симв + длина2 - 1], успех)
```

В программе на языке Паскаль можно также после пропуска слов и пробелов считать оставшиеся символы-цифры как два отдельных числа:

```
readln(chislo1, chislo2);
```

Если длина чисел известна, то необходимые значения можно получить, зная их “местонахождение” в конце строки.

1.8. Выделение трех чисел после второго слова

Такая задача, используемая, например, в [3], решается аналогично.

Задания для самостоятельной работы учащихся

1. Задана строка с информацией об ученике в формате: <Фамилия> <Номер школы>
— где <Фамилия> — значение, состоящее не более чем из 30 символов без пробелов, < Номер школы> — целое число в диапазоне от 1 до 99. Эти данные записаны через один пробел (то есть всего в строке один пробел).

Получить фамилию ученика:

1) включая пробел после нее;

2) без указанного пробела.

Задачу 2) решить также для случая, когда длина величины < Фамилия > известна.

2. Задана строка с информацией о стоимости бензина на АЗС в формате:

<Компания> <Улица> <Цена>

— где <Компания> — строка, состоящая не более чем из 20 символов без пробелов, <Улица> — строка, состоящая не более чем из 20 символов без пробелов, <Марка> — одно из чисел 92, 95 или 98, <Цена> — целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках. <Компания> и <Улица>, <Улица> и <Цена> разделены ровно одним пробелом (то есть всего два пробела).

Получить название улицы, на которой находится данная АЗС.

Задачу решить двумя способами:

1) с использованием оператора цикла;

2) без его использования.

3. Задана строка с информацией об ученике в формате:

<Фамилия > <Имя> <Оценка>

— где <Фамилия> — значение, состоящее не более чем из 30 символов без пробелов, <Имя> — строка, состоящая не более чем из 20 символов без пробелов, <Оценка> — целое число в диапазоне от 2 до 5. Эти данные записаны через один пробел, причем ровно один между каждой парой (то есть всего два пробела в строке).

Получить фамилию и имя ученика как одну величину:

1) включая пробелы до и после имени;

2) без пробела после имени.

4. Задана строка с информацией о футбольной команде в формате:

<Название> <Количество>

— где <Название> — значение, состоящее не более чем из 20 символов без пробелов, <Количество> — число раз, которое команда участвовала в чемпионате. Эти данные записаны через один пробел (то есть всего в строке один пробел).

Определить, сколько раз данная команда участвовала в чемпионате.

5. Задана строка с информацией об ученике в формате:

<Название города> <Год основания>

— где <Название города> — значение, состоящее не более чем из 20 символов без пробелов, <Год основания> — четырехзначное число. Эти значения записаны через один пробел (то есть всего в строке один пробел).

Определить год основания данного города. Задачу решить без использования оператора цикла.

6. Задана строка с информацией об ученике в формате:

<Фамилия > <Имя> <Номер школы>

— где <Фамилия> — значение, состоящее не более чем из 30 символов без пробелов, <Имя> — строка, состоящая не более чем из 20 символов без пробелов, <Номер школы> — целое число в диапазоне от 1 до 1599. Эти данные записаны через один пробел, причем ровно один между каждой парой (то есть всего два пробела в строке).

Определить номер школы.

7. Задана строка с информацией об ученике в формате:

<Фамилия> <Имя> <Год рождения>

— где <Фамилия> — значение, состоящее не более чем из 20 символов без пробелов, <Имя> — строка, состоящая не более чем из 15 символов без пробелов, <Год рождения> — четырехзначное число. Эти значения записаны через один пробел (то есть всего в строке два пробела).

Определить год рождения данного ученика. Задачу решить без использования оператора цикла.

8. Задана строка с информацией о марке и стоимости бензина на АЗС в формате:

<Компания> <Улица> <Марка> <Цена>

— где <Компания> — строка, состоящая не более чем из 20 символов без пробелов, <Улица> — строка, состоящая не более чем из 20 символов без пробелов, <Марка> — одно из чисел 92, 95 или 98, <Цена> — целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках. <Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <Цена> разделены ровно одним пробелом (то есть всего в строке три пробела).

Получить марку и стоимость бензина.

9. Задана строка с информацией об оценках ученика по пятибалльной системе по трем предметам в формате:

<Фамилия> <Имя> <Оц1> <Оц2> <Оц3>

— где каждое значение отделено друг от друга одним пробелом (то есть всего в строке четыре пробела).

Определить каждую из оценок.

Задачу решить двумя способами:

1) с использованием оператора цикла;

2) без его использования.

2. Группа задач на подсчет количества каждого из значений

2.1. Подсчет количества цифр в заданной последовательности

Задача: “Даны *n* цифр. Подсчитать количество каждой из них. Массив для хранения цифр заданной последовательности не использовать”.

Анализ решения

Прежде всего, учитывая требование, связанное с массивом цифр, каждую цифру последовательности следует “обрабатывать” сразу после ее ввода. Это же надо делать и с другими аналогичными данными в задачах, рассматриваемых ниже.

Далее, получать искомое количество каждой из цифр с помощью 10 переменных величин нерационально — лучше использовать массив *кол_цифр* из 10 элементов с индексами от 0 до 9.

Обсудим, как учесть очередную цифру в этом массиве.

Проверка каждой из цифр с использованием оператора выбора (варианта) производится следующим образом:

```
выбор
  при цифра = 0:
  |Увеличиваем количество цифр 0
  кол_цифр[0] := кол_цифр[0] + 1
  при массив1[i] = 1:
  |Увеличиваем количество цифр 1
  кол_цифр[1] := кол_цифр[1] + 1
  …
  при массив1[i] = 9:
  |Увеличиваем количество цифр 9
  кол_цифр[9] := кол_цифр[9] + 1
 все
кц
```

— также является нерациональной.

Подумаем, значение какого элемента массива *кол_цифр* должно быть увеличено на 1, если очередная цифра равна *цифра*. Ответ — элемента с

индексом *цифра*. Этот позволяет записать оператор для подсчета количества каждой из цифр в последовательности очень компактно:

```
…
ввод n
нц для i от 1 до n
 |Вводим очередную цифру
 ввод цифра
 |Увеличиваем ее "счетчик" на 1
 кол_цифр[цифра] := кол_цифр[цифра] + 1
кц
```

Язык Паскаль

```
…
{Рассматриваем все заданные цифры}
for i := 1 to n do
 begin
  {Вводим очередную цифру}
  read(zifra);
  {Увеличиваем ее "счетчик" на 1}
  Inc(kol_zifr[zifra])
 end;
```

Примечание. В школьном алгоритмическом языке начальное присваивание элементам массива нулевых значений не происходит, поэтому следует предварительно обнулить все элементы массива *кол_цифр*. В программах на языке Паскаль и ряде других это не является обязательным. Вместе с тем, "правилом хорошего тона" является начальное присваивание переменным величинам нулевого значения. (См. также обсуждение вопросов, связанных с эффективностью программ, в начале статьи.)

2.2. Подсчет количества цифр в заданной строке

Вариант 1

Задача: "Дана строка, состоящая из цифр. Подсчитать количество каждой из цифр. Массив для хранения отдельных цифр заданной строки не использовать".

Анализ решения

Здесь, как и в предыдущей задаче, искомое количество каждой из цифр запишем в массив *кол_цифр* из 10 элементов с индексами от 0 до 9.

Идея решения задачи достаточно понятная:

1) выделить отдельную цифру строки в виде числа (пусть ее имя — *цифра*);

2) учесть ее в массиве *кол_цифр*.

Соответствующий фрагмент:

```
…
|Ввод строки
ввод строка
|Обнуляем элементы массива кол_цифр
нц для i от 0 до 9
 кол_цифр[i] := 0
кц
|Рассматриваем все символы заданной строки
нц для i от 1 до длин(строка)
 |Преобразуем i-й символ в число
 цифра := лит_в_цел(строка[i], успех)
 |Увеличиваем счетчик этой цифры на 1
 кол_цифр[цифра] := кол_цифр[цифра] + 1
кц
```

Язык Паскаль

```
{Обнуляем элементы массива kol_zifr}
for i := 0 to 9 do kol_zifr[i] := 0;
{Рассматриваем все символы
 заданной строки}
for i := 1 to Length(stroka) do
 begin
  {Преобразуем символ-цифру stroka[i]
   в число}
  Val(stroka[i], zifra, code);
  {Учитываем его в массиве kol_zifr}
  Inc(kol_zifr[zifra])
 end;
```

Программа на языке Паскаль может быть упрощена за счет того, что значениями индексов элементов массива `kol_zifr` в этом языке могут быть данные символьного типа. Это позволит отказаться от преобразования символов в число:

```
var
kol_zifr: array ['0'..'9'] of byte;
…
BEGIN
{Ввод строки}
readln(stroka);
{Рассматриваем все символы заданной
 строки}
for i := 1 to Length(stroka) do
  {Для символа-цифры stroka[i]
   увеличиваем на 1
  его значение в массиве kol_zifr}
  Inc(kol_zifr[stroka[i]]);
…
```

Как отмечалось в разделе 1, в программе на языке Паскаль можно отдельно не вводить всю заданную строку и потом обрабатывать ее, а выделять символы-цифры сразу во время ввода исходных данных. Это можно сделать благодаря использованию оператора `read`, аргументом которого является величина символьного типа (`char`):

```
var
 dlina: integer;  simv: char; …
BEGIN
…
write('Задайте длину строки ');
read(dlina);
{Считываем по одному символу
 заданной строки}
i := 1;
writeln('Введите строку ');
repeat
 read(simv);
 {Для очередного символа  увеличиваем
  на 1 его значение в массиве kol_zifr}
 Inc(kol_zifr[simv]);
 {Переходим к следующему символу}
 i := i + 1
until i > dlina;
```

Аналогично на языке Паскаль можно решать задачи, в которых известна не длина строки, а ее последний символ. В этом случае величина *dlina* не используется:

```
var
  simv: char; …
BEGIN
…
writeln('Введите строку ');
{Считываем первый символ}
read(simv);
while simv <> posled do
 begin
  {Для очередного символа
  увеличиваем на 1 его значение
  в массиве kol_zifr}
  Inc(kol_zifr[simv]);
  {Считываем следующий символ}
  read(simv);
 end;
```

— где *posled* — последний символ строки.

Заметим, что применение оператора цикла с постусловием:

```
writeln('Введите строку ');
repeat
 {Считываем очередной символ}
 read(simv);
 {Для очередного символа
 увеличиваем на 1 его значение
 в массиве kol_zifr}
 Inc(kol_zifr[simv]);
until simv = posled;
```

— в данном случае невозможно, так как для последнего символа — "нецифры" нельзя выполнить процедуру Inc.

Вариант 2

Задача: "Дана строка, в которой есть цифры. Подсчитать количество каждой из цифр. Массив для хранения отдельных цифр заданной строки не использовать".

Анализ решения

Здесь новым является то, что в заданной строке, кроме цифр, имеются и другие символы, и наша задача — выделить и обработать только символы-цифры. Выделение цифр можно провести так:

```
|Ввод строки
…
|Рассматриваем все символы заданной строки
нц для i от 1 до длин(строка)
  |Если i-й символ - цифра
  если строка[i] >= "0" и строка[i] <= "9"
    то
     |Преобразуем его в число
     цифра := лит_в_цел(строка[i], успех)
     |Учитываем цифру в массиве кол_цифр
     кол_цифр[цифра] := кол_цифр[цифра] + 1
  все
кц
```

Язык Паскаль

```
for i := 1 to Length(stroka) do
  if (simv >= '0') and (simv <= '9')
  {Если i-й символ - цифра}
    then  Inc(kol_zifr[stroka[i]]);
```

Задачи такого типа, в которых известна не длина строки, а ее последний символ, на языке Паскаль можно решать так же, как и предыдущую задачу:

```
writeln('Введите строку ');
{Считываем первый символ}
read(simv);
while simv <> posled do
 begin
   if (simv >= '0') and (simv <= '9')
   then Inc(kol_zifr[simv]);
  {Считываем следующий символ}
  read(simv);
 end;
```

— где *posled* — последний символ строки.

Интересно, что в этой задаче применение оператора цикла с постусловием возможно:

```
writeln('Введите строку ');
repeat
 {Считываем очередной символ}
 read(simv);
  if (simv >= '0') and (simv <= '9')
  then Inc(kol_zifr[simv]);
until simv = posled;
```

2.3. Подсчет количества букв в заданной строке

Вариант 1

Задача: "Дана строка, состоящая из прописных букв латинского алфавита. Подсчитать количество каждой из букв алфавита. Массив для хранения отдельных букв заданной строки не использовать".

Анализ решения

В программе на Паскале задача решается во многом аналогично первому варианту задачи 2.2, если использовать массив kol_bykv с индексами от "A" до "Z":

```
var
kol_bykv: array ['A'..'Z'] of byte;
…
BEGIN
{Ввод строки}
Readln(stroka);
{Рассматриваем все буквы заданной строки}
for i := 1 to Length(stroka) do
  {Для буквы stroka[i] увеличиваем на 1
  ее значение в массиве kol_bykv }
  Inc(kol_bykv[massiv1[i]]);
…
```

В программе на школьном алгоритмическом языке заполняемый массив *кол_букв* должен иметь индексы, соответствующие кодам прописных букв латинского алфавита:

```
цел таб кол_букв[65:90]
```

— а в программе должны использоваться коды букв:

```
...
|Ввод строки
ввод строка
|Обнуляем элементы массива кол_букв
нц для i от 65 до 90
 кол_букв[i] := 0
кц
|Рассматриваем все буквы заданной строки
нц для i от 1 до длин(строка)
 |Выделяем i-ю букву
 буква := строка[i]
 |Увеличиваем счетчик этой буквы на 1
 кол_букв[код(буква)] :=
                кол_букв[код(буква)] + 1
кц
```

Вариант 2

Задача: "Дана строка, в которой есть прописные буквы латинского алфавита. Подсчитать количество каждой из букв алфавита. Массив для хранения отдельных букв не использовать".

Анализ решения

Здесь, как и в аналогичной задаче 2.2, особенность в том, что в заданной строке, кроме прописных букв латинского алфавита, имеются и другие символы, и наша задача — выделить и обработать только нужные буквы. Это можно сделать так:

```
|Ввод строки
...
|Рассматриваем все символы заданной строки
нц для i от 1 до длин(строка)
  |Если i-й символ - прописная буква
  |латинского алфавита
  если строка[i] >= "A" и
       строка[i] <= "Z"
    то
      ...
```

Примечание. Если в двух последних задачах предполагается, что в строке есть строчные буквы, а искомое количество каждой из букв необходимо найти без учета регистра, то в программах на Паскале можно использовать функцию UpCase.

2.4. Подсчет количества каждого из числовых значений в заданной последовательности чисел

Задача: "Даны *n* чисел со значениями от 0 до 50. Подсчитать количество каждой из них. Массив для хранения чисел заданной последовательности не использовать".

Задача решается аналогично задаче 2.1 (используется массив *кол_чисел/kol_chisel* из 51 элемента).

2.5. Подсчет количества каждого из числовых значений в заданном наборе строк

Задача: "Даны *n* строк, в каждой из которых находится информация в формате:

<Подстрока> <Число>

— где <Подстрока> — последовательность из не более чем 30 букв без пробелов, <Число> — целое число в диапазоне от 1 до 99. Эти данные записаны через пробел (то есть всего один пробел в каждой строке).

Подсчитать количество каждого из чисел".

Анализ решения

Общая схема решения задачи заполнения массива с количествами каждого из значений от 1 до 99:

```
1. Ввод значения n
2. Цикл для i от 1 до n
  2.1. Ввод i-й строки
  2.2. Выделение числа
  2.3. Учет этого числа в массиве "счет-
чиков"
  конец цикла
```

Возможные методы выполнения этапа 2.2 рассмотрены в разделе 1. Учет числа в массиве "счетчиков" проводится так же, как и в задачах 2.1–2.4 (естественно, с учетом диапазона значений).

Здесь же заметим, что в программе на Паскале использование массива с индексами строкового типа от "1" до "99" недопустимо.

Задания для самостоятельной работы учащихся

1. Известны оценки каждого из 25 учеников класса на контрольной работе по физике. Подсчитать количество двоек, троек, четверок и пятерок. Массив для хранения оценок и четыре переменные для хранения искомых значений не использовать.

2. Дана строка, состоящая из цифр 3, 4, 5, 6, 7. Подсчитать количество каждой из таких цифр. Массив для хранения отдельных цифр заданной строки и пять переменных для хранения искомых значений не использовать.

3. Дана строка, в которой есть числовые значения. Подсчитать количество каждой из цифр. Массив для хранения отдельных цифр заданной строки не использовать.

4. Дан текст, состоящий из не более чем 250 прописных букв латинского алфавита. Подсчитать количество каждой из букв алфавита. Массив для хранения отдельных букв заданной строки не использовать.

5. Дан текст, состоящий из не более чем 250 букв латинского алфавита. Подсчитать количество каждой из прописных букв алфавита. Массив для хранения отдельных букв заданной строки не использовать.

6. Даны *n* строк, в каждой из которых находится информация в формате:

<Фамилия> <Номер школы>

— где <Фамилия> — строка, состоящая не более чем из 30 символов без пробелов, <Номер школы> — целое число в диапазоне от 1 до 99. Эти данные записаны через пробел (то есть всего один пробел в каждой строке).

Подсчитать количество учеников в каждой из школ. Массив для хранения номеров школ не использовать.

3. Группа задач на подсчет количества и вывод значений, удовлетворяющих некоторому условию

3.1. Подсчет количества тех чисел последовательности, которые удовлетворяют некоторому условию

Задача в общем виде: "Даны *n* чисел. Определить количество чисел, удовлетворяющих некоторому условию".

Анализ решения

В приведенном далее фрагменте решения задачи <условие> — заданное условие для подсчета чисел (это может быть равенство, неравенство, сложное условие и т.п.), *количество* — искомое количество:

```
количество := 0
нц для i от 1 до n
  |Ввод очередного числа a
  …
  |Если заданное условие соблюдается
  если <условие>
    то
     |Учитываем число a в искомом количестве
     количество := количество + 1
  все
кц
```

Заметим, что <условие> зависит от:

1) значения числа *a*;

2) значения числа *a* и от порядкового номера *i*.

Если оно зависит только от порядкового номера *i*, то задача может быть решена без использования оператора цикла (убедитесь в этом!).

В случае, если обрабатываемое число является частью строки, необходимо предварительно извлечь нужное значение из строки (см. типовые задачи группы 1).

Применительно к массиву чисел задача решается аналогично.

Язык Паскаль

```
kolichestvo := 0;
for i := 1 to n do
 begin
  {Ввод очередного числа a}
  …
  {Если заданное условие соблюдается|
  if <условие>
    then
     {Учитываем число a в искомом
      количестве}
     kolichestvo := kolichestvo + 1
     {или inc(kolichestvo)}
end
```

3.2. Вывод на экран элементов массива с заданными свойствами

Задача в общем виде: "Дан массив. Вывести на экран его элементы, удовлетворяющие некоторому условию. Известно, что элементы с заданными свойствами в массиве имеются".

Анализ решения

Соответствующий фрагмент:

```
нц для i от 1 до n
  |Если очередной элемент обладает
  |заданными свойствами
  если <условие>
    то
     |Выводим его
     вывод a[i], " "
  все
кц
```

Об особенностях значения <условие> — см. задачу 3.1.

Вариант 2 задачи: "Дан массив. Вывести на экран те его элементы, которые удовлетворяют некоторому условию, а затем на отдельной строке — количество таких элементов. Известно, что в массиве есть как минимум один элемент с заданными свойствами".

Анализ решения

Вспомнив методику решения задачи 3.1, можем записать:

```
количество := 0
нц для i от 1 до n
  |Если очередной элемент обладает
  |заданными свойствами
  если <условие>
    то
     |Выводим его значение
     вывод  a[i], " "
     |и учитываем его в искомом
     |количестве
     количество := количество + 1
  все
кц
|Выводим искомое количество
вывод нс, количество
```

Вариант 3 задачи: "Дан массив. Вывести на экран те его элементы, которые удовлетворяют некоторому условию. Если таких элементов нет, вывести на экран сообщение «Таких значений нет»".

Задача решается аналогично варианту 2 — изменения должны коснуться вывода сообщения вместо количества значений:

```
…
нц для i от 1 до n
  |Если очередной элемент обладает
  |заданными свойствами
  если <условие>
    то
     вывод  a[i], " "
     количество := количество + 1
  все
кц
```

```
если количество = 0
 то
  |Выводим сообщение
  вывод нс, "Таких значений нет"
все
```

Вариант 4 задачи: “Дан массив. Вывести на экран те его элементы, которые удовлетворяют некоторому условию. Если такой элемент единственный, то на следующей строке вывести его индекс. Известно, что указанные элементы в массиве имеются”.

Здесь надо также запоминать индекс элемента, удовлетворяющего заданному условию. Если он окажется единственным, то именно его индекс и будет выведен, в противном случае — индекс не понадобится:

```
...
нц для i от 1 до n
  |Если очередной элемент обладает
  |заданными свойствами
  если <условие>
    то
     вывод i, " " |Выводим его
     |Запоминаем его индекс
     индекс := i
     |и учитываем его в количестве
     количество := количество + 1
  все
кц
если количество = 1
 то
  |Выводим его индекс
  вывод нс, индекс
все
```

3.3. Вывод на экран индексов элементов массива с заданными свойствами

Задача в общем виде: “Дан массив. Вывести на экран индексы тех его элементов, которые удовлетворяют некоторому условию. Известно, что элементы с заданными свойствами в массиве имеются”.

Задача решается аналогично варианту 1 предыдущей задачи (выводятся не значения, а индексы соответствующих элементов).

Вариант 2 задачи: “Дан массив. Вывести на экран индексы тех его элементов, которые удовлетворяют некоторому условию, а затем на отдельной строке — количество таких элементов. Известно, что в массиве есть как минимум один элемент с заданными свойствами”.

Задача решается аналогично варианту 2 задачи 3.2.

Вариант 3 задачи: “Дан массив. Вывести на экран индексы тех его элементов, которые удовлетворяют некоторому условию. Если таких элементов нет, вывести на экран сообщение «Таких значений нет»”.

Задача решается аналогично варианту 3 задачи 3.2.

Вариант 4 задачи: “Дан массив. Вывести на экран индексы тех его элементов, которые удовлетворяют некоторому условию. Если такой элемент единственный, то на следующей строке вывести его значение. Известно, что указанные элементы в массиве имеются”.

Задача решается аналогично варианту 4 задачи 3.2.

В заключение обратим внимание на одно важное обстоятельство. Если в условии, которое при обсуждении рассмотренных задач было обозначено как <условие>, должно использоваться равенство с данными вещественного типа, значения которых рассчитываются, то его необходимо оформлять как неравенство с учетом точности вычислений с такими данными. Например, если сравниваются два значения a и b, рассчитанные с точностью до одной цифры после десятичной точки, то условие следует оформить так:

$$\text{abs}(a - b) < 0.0001$$

— где abs — функция, возвращающая абсолютную величину ее аргумента.

3.4. Вывод на экран элементов массива, соответствующих элементам другого массива с заданными свойствами

Задача в общем виде: “Даны два массива. Вывести на экран те элементы первого массива, которые соответствуют элементам второго массива с заданными свойствами”.

Пример задачи “Даны два массива с информацией о 25 людях:

1) с их фамилиями;

2) со значениями их роста в см.

Вывести на экран фамилии людей, рост которых превышает 170 см. Известно, что такие люди среди представленных имеются”.

Решение в общем виде

Соответствующий фрагмент:

```
нц для i от 1 до n
  |Если очередной элемент обладает заданными
  |свойствами
  если <условие, связанное со вторым
        массивом>
    то
     |Выводим соответствующий элемент
     |1-го массива
     вывод массив1[i], " "
  все
кц
```

Об особенностях значения <условие> — см. задачу 3.1.

Решение для приведенной задачи

```
нц для i от 1 до n
   если рост[i] > 170
    то
     вывод фамилия [i], " "
   все
кц
```

— где *рост* и *фамилия* — заданные массивы

Примечание. При решении задачи такого типа с использованием языка Паскаль можно вместо двух массивов использовать один массив с данными типа “запись”.

В заключение заметим, что методика решения задач на подсчет количества максимальных/минимальных значений рассмотрена в разделе 4.

Задания для самостоятельной работы учащихся

1. Известны оценки 22 учеников класса по алгебре. Определить количество четверок.

2. Дан массив целых чисел из 12 элементов. Определить количество чисел, оканчивающихся нулем (известно, что такие числа в массиве есть).

3. Дан массив натуральных чисел из 10 элементов. Вывести на экран те из них, которые кратны трем, а на следующей строке — количество таких чисел. Известно, что числа, о которых идет речь, в массиве есть.

4. В массиве записана информация о росте каждого из 20 человек (в см). Вывести значения роста, большие 190 см. Если таких значений в массиве нет, вывести сообщение “Нет людей с таким ростом”.

5. Дан массив из 15 элементов с вещественными значениями. Вывести отрицательные элементы. Если такой элемент единственный, то на следующей строке вывести его индекс. Известно, что в массиве есть как минимум один отрицательный элемент.

6. В массиве записано количество участников олимпиады из каждой из школ с номерами от 10 до 20. Вывести номера школ, число участников из которых больше двух, а затем на отдельной строке — количество таких школ.

7. В массиве записана информация о количестве баллов, набранных каждым из 20 учащихся на ЕГЭ по информатике. Вывести условные номера учащихся, набравших более 90 баллов (условный номер соответствует порядковому номеру баллов учащихся в массиве). Если таких учащихся нет, вывести сообщение “Нет таких учащихся”.

8. Дан массив с общим количеством учащихся в каждой из параллелей школы (от 1-й до 11-й). Вывести номера параллелей, общее число учащихся в которых превышает 60. Если такая параллель — единственная, то на следующей строке вывести число учащихся в ней. Известно, что в массиве есть как минимум одна параллель с указанным числом учащихся.

9. В массиве записаны значения роста 15 человек в формате xxx.xx, где x — цифра. Определить число людей, рост которых равен среднему значению роста.

10. Даны два массива с информацией о 30 участниках олимпиады по информатике:

1) с их фамилиями;

2) с суммой набранных ими баллов.

Вывести на экран фамилии участников, набравших не менее 30 баллов. Известно, что такие учащиеся имеются.

4. Группа задач на нахождение максимальных (минимальных) элементов массива, их индексов, количеств и т.п.

Хотя методика решения всех задач этой группы описана применительно к массивам, подобные задачи для последовательности чисел в большинстве случаев решаются аналогичными методами (вместо *i*-го элемента массива нужно рассматривать очередное число последовательности).

При анализе предполагается, что массив имеет размер в *n* элементов.

4.1. Определение максимального элемента массива

Алгоритм решения этой задачи аналогичен алгоритму действий человека, который определяет максимальное значение в некоторой одномерной таблице с числами. Сначала он смотрит в первую ячейку таблицы и запоминает записанное там число. Затем смотрит во вторую ячейку и в случае, если имеющееся там число больше запомненного, в качестве максимального запоминает новое число. Для остальных ячеек действия аналогичны.

Соответствующий фрагмент программы:

```
|Начальное присваивание значения искомой
|величине
максимальное := a[1]
|Рассматриваем остальные элементы
нц для i от 2 до n
  |Сравниваем i-й элемент со значением
  |максимальное
  если a[i] > максимальное
    то
      |Принимаем встреченный элемент
      |в качестве максимальное
      максимальное := a[i]
  все
кц
```

Если диапазон возможных значений элементов в массиве известен, то фрагмент программы решения задачи может быть оформлен так:

```
максимальное := минимальное
|Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > максимальное
    то
      максимальное := a[i]
  все
кц
```

— где *минимальное* — нижняя граница диапазона возможных значений.

Так, например, если известно, что в массиве все элементы неотрицательные, то можем записать:

```
максимальное := 0
нц для i от 1 до n
  если a[i] > максимальное
    то
      максимальное := a[i]
  все
кц
```

4.2. Определение минимального элемента массива

Задача решается аналогично предыдущей (конечно, с необходимыми изменениями).

4.3. Определение индекса максимального элемента массива

Здесь также алгоритм решения задачи аналогичен алгоритму действий человека, определяющего номер ячейки с максимальным значением в некоторой одномерной таблице с числами — сначала он запоминает первое число и номер 1, а затем рассматривает второе число. Если оно больше того числа, которое помнил, то запоминает новое число и номер 2 и переходит к следующему, в противном случае просто переходит к следующему, третьему, числу и делает то же самое и т.д.

Фрагмент программы, в котором решается задача:

```
|Начальное присваивание значений
|искомым величинам
максимальное := a[1]
номер_максимального := 1
|Рассматриваем остальные элементы
нц для i от 2 до n
  если a[i] > максимальное
    то
      |Принимаем встреченный элемент
      |в качестве максимальное
      максимальное := a[i]
      |а его индекс — в качестве
      |номер_максимального
      номер_максимального := i
  все
кц
```

Возникает вопрос — индекс какого элемента будет найден, если в массиве есть несколько элементов с максимальным значением? Что надо изменить в приведенном фрагменте, чтобы находился индекс последнего элемента с таким значением?

Если диапазон возможных значений элементов массива известен, то можно оформить фрагмент так:

```
максимальное := минимальное[1]
|Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > максимальное
    то
      |Принимаем встреченный элемент
      |в качестве максимальное
      максимальное := a[i]
      |а его индекс — в качестве
      |номер_максимального
      номер_максимального := i
  все
кц
```

— где *минимальное* — нижняя граница диапазона возможных значений.

Видно, что в результате определяются два значения — *номер_максимального* и *максимальное*.

Если последнее значение находить не требуется, то без этой величины можно вообще обойтись. В самом деле, если нам известно значение индекса максимального среди рассмотренных элементов, то мы знаем и значение соответствующего элемента (оно равно *a*[*номер_максимального*]):

```
номер_максимального := 1
нц для i от 2 до n
  если a[i] > a[номер_максимального]
    то
      номер_максимального := i
  все
кц
```

4.4. Нахождение индекса минимального элемента

Задача решается способами, аналогичными описанным применительно к предыдущей задаче.

4.5. Нахождение второго по величине максимального элемента

Данная задача допускает два толкования. Если рассматривать, например, массив 5 10 22 6 22 20 6 12, то каким должен быть ответ?

Под “вторым по величине максимальным элементом”, или, короче, “вторым максимумом”, можно понимать:

1) значение элемента массива, который стоял бы на предпоследнем месте, если бы массив был отсортирован по неубыванию. При таком толковании — 22;

2) значение элемента массива, *больше* которого только максимальный. В этом случае ответ — 20.

Если в массиве только один максимальный элемент (все остальные меньше), то оба толкования совпадают, и искомые значения будут одними и теми же, в противном случае — нет.

Обсудим оба варианта задачи. В каждом из них будем использовать две переменные:

1) *максимум*1 — максимальный элемент массива (первый максимум);

2) *максимум*2 — второй максимум (искомое значение).

4.5.1. Поиск элемента массива, который стоял бы на предпоследнем месте, если бы массив был отсортирован по неубыванию

Сначала рассмотрим вариант, в котором диапазон значений элементов массива известен (а именно это имеет место во всех задачах, связанных с ЕГЭ).

В качестве начальных значений величин *максимум*1 и *максимум*2 (см. выше) принимаем число, которое заведомо меньше нижней границы диапазона значений элементов массива (например, при диапазоне от –1000 до 1000 — число –1001):

```
максимум1 := -1001
максимум2 := -1001
```

Строго говоря, значение *максимум*2 можно не задавать, что будет показано ниже.

---

[1] Если допускается, что все элементы массива могут быть равны значению *минимальное*, то начальное присваивание должно быть таким: `максимальное := минимальное - 1`.

Затем рассматриваем все элементы, сравнивая их сначала со значением *максимум*1, а затем (при необходимости) — и со значением *максимум*2:

```
нц для i от 1 до n
  если a[i] > максимум1
    |Встретился элемент, больший максимум1
    то
     |Бывший первый максимум станет вторым
     максимум2 := максимум1
     |Первым максимумом станет
     |встреченный элемент
     максимум1 := a[i]
     |Внимание - именно в таком порядке!
    иначе
     |Очередной элемент не больше максимум1
     |Сравниваем его со значением максимум2
     если a[i] > максимум2
     |Только в этом случае!
     |Встретился элемент, больший максимум2
       то
        |Принимаем его в качестве
        |нового значения максимум2
        максимум2 := a[i]
        |Значение максимум1 не меняется
     все
 все
кц
```

Нетрудно убедиться, что уже после рассмотрения первого элемента произойдет “переприсваивание”

```
максимум2 := максимум1
```

— то есть начальное значение величины *максимум*2, равное –1001, можно не задавать.

Если же диапазон значений элементов массива неизвестен, то предварительно нужно определить начальные значения величин *максимум*1 и *максимум*2, сравнив первый и второй элементы массива:

```
если a[2] > a[1]
  то
    максимум1 := a[1]
    максимум2 := a[2]
  иначе
    максимум1 := a[2]
    максимум2 := a[1]
все
```

или, короче:

```
максимум1 := a[1]
максимум2 := a[2]
если a[2] > a[1]
  то
   максимум1 := a[2]
   максимум2 := a[1]
все
```

Затем рассматриваем остальные элементы, как и ранее, сравнивая их сначала со значением *максимум*1, а затем (при необходимости) — и со значением *максимум*2:

```
  нц для i от 3 до n
    если a[i] > максимум1
      ...
```

4.5.2. Нахождение элемента массива, больше которого только максимальный

Примем, что диапазон значений элементов массива известен.

В качестве начального значения величины *максимум*1[2] принимаем число, которое заведомо меньше нижней границы диапазона значений элементов массива (например, при диапазоне от –1000 до 1000 — число –1001):

```
максимум1 := -1001
```

Рассматриваем все элементы массива и проверяем каждое из них сначала на первый, а затем на второй максимум:

```
нц для i от 1 до n
  если a[i] > максимум1
    |Встретился элемент, больший максимум1
    то
     |Бывший первый максимум станет вторым
     максимум2 := максимум1
     |Первым максимумом станет встреченный
     |элемент
     максимум1 := a[i]
    иначе
     |Очередной элемент не больше максимум1.
     |В этом случае
     |сравниваем его со значением максимум2
     если a[i] < максимум1 и
          a[i] > максимум2
       |Встретился элемент, меньший
       |максимум1 и больший максимум2
       то
        |Принимаем его в качестве
        |нового значения максимум2
        максимум2 := a[i]
        |Значение максимум1 не меняется
     все
 все
кц
```

Фрагмент программы, относящийся к выводу ответа, оформляется так:

```
если максимум2 = -1001
  |Второй максимум не встретился
  то
   вывод нс, "Нет такого значения в массиве "
  иначе
   вывод нс, "Второй максимум равен ",
             максимум2
все
```

Ясно, что второго максимума в принятом толковании может не быть только тогда, когда все элементы массива равны.

4.6. Нахождение второго минимума

Все варианты этой задачи решаются аналогично предыдущей (конечно, с необходимыми изменениями).

---

[2] Для величины максимум2 начальное значение, как и для первого варианта, можно не задавать.

4.7. Нахождение количества максимальных элементов

Задача может быть решена двумя способами:

1) за два прохода по массиву;

2) за один проход по массиву.

В первом случае решение очевидно — на первом проходе следует найти максимальный элемент массива (см. задачу 4.1), а на втором — подсчитать количество элементов, равных максимальному значению (см. задачу 3.1).

Во втором случае идея решения такая: проходя по массиву, кроме значения максимума, контролировать также количество элементов, равных максимальному (*кол_макс*). Если очередной элемент оказывается больше текущего максимума — он принимается в качестве максимального значения, а величина *кол_макс* — равной 1. Если же очередной элемент не больше максимального, то сравниваем его с максимумом. Если они равны, то встретился еще один максимум, и значение *кол_макс* увеличиваем на 1.

```
|Начальное присваивание значений
|искомым величинам
максимальное := a[1]
кол_макс := 1
|Рассматриваем остальные элементы
нц для i от 2 до n
  если a[i] > максимальное
    то      |Встретился новый максимум
     |Принимаем его в качестве значения
     |максимальное
     максимальное := a[i]
     |Пока он — единственный
     кол_макс := 1
    иначе
     |Проверяем, не равно ли очередное
     |значение "старому" максимуму
     если a[i] = максимальное
       то  |Встретился еще один максимум
           |Учитываем это
           кол_макс := кол_макс + 1
    все
  все
кц
```

Если диапазон возможных значений элементов массива известен, то и здесь можно отдельно не рассматривать первый элемент:

```
|Начальное присваивание значений
|искомым величинам
максимальное := минимальное - 1
|Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > максимальное
...
```

— где *минимальное* — нижняя граница диапазона возможных значений.

В задачах, в которых количество максимальных элементов выводится на экран после вывода их индексов, целесообразно подсчет этого количества проводить во время второго прохода (при отборе нужных элементов и выводе индексов):

```
|1-й проход — определение максимального
|значения
...
|2-й проход — отбор элементов
|с максимальным значением
|и вывод их индексов
кол_макс := 0
|Рассматриваем все элементы
нц для i от 1 до n
  если a[i] = максимальное
   то
    |Вывод индекса
    вывод i, " "
    кол_макс := кол_макс + 1
  все
кон
|Вывод значения кол_макс
...
```

4.8. Нахождение количества минимальных элементов

Задача решается аналогично предыдущей (конечно, с необходимыми изменениями).

4.9. Нахождение количества вторых максимумов

Решение задачи также зависит от толкования понятия “второй максимум” (см. задачу 4.6). Начнем со второго толкования.

4.9.1. Нахождение количества значений в массиве, больше которых только максимальный элемент массива

Примем, что диапазон значений элементов массива известен (а именно это имеет место во всех задачах, связанных с ЕГЭ).

В качестве начального значения величины *максимум*1 (см. выше) принимаем число, которое заведомо меньше нижней границы диапазона значений элементов массива (например, при диапазоне от –1000 до 1000 — число –1001):

```
максимум1 := -1001
```

Значение *максимум*2, как и при решении задачи 4.5.2, можно не задавать.

Обсудим вопрос о начальных значениях искомых величин *кол_максимум*1 и *кол_максимум*2 (их смысл ясен из имен).

Значение *кол_максимум*2 также можно не задавать. Если вспомнить (см. задачу 4.5.2), как происходит “переприсваивание” значений, то можно сказать, что уже после рассмотрения первого элемента произойдет “переприсваивание” не только значения, но и

```
максимум2 := максимум1
кол_максимум2 := кол_максимум1
```

Значение же *кол_максимум*1 также можно задать любым “неположительным” (например, 0). После

рассмотрения первого элемента оно “перейдет” ко второму максимуму, а после “встречи” первого элемента, меньшего максимального — значение *кол_максимум*2 станет равным 1 (убедитесь в этом, рассмотрев возможные варианты согласно приведенному далее фрагменту программы). Если же все элементы массива равны, то это значение *кол_максимум*2 останется неизменным.

Итак, после задания начальных значений *максимум*1 и *кол_максимум*1 рассматриваем все элементы массива, сравнивая их сначала со значением *максимум*1, а затем (при необходимости) — и со значением *максимум*2:

```
нц для i от 1 до n
 если a[i] > максимум1
  |Встретился элемент, больший максимум1
  то
   |Бывший первый максимум станет вторым
   максимум2 := максимум1
   |а значение кол_максимум2 станет равно
   |"старому" значению величины
   |кол_максимум1
   кол_максимум2 := кол_максимум1
   |Первым максимумом станет встреченный
   |элемент
   максимум1 := a[i]
   |Внимание - именно в таком порядке!
   |Новый первый максимум встретился
   |впервые,
   кол_максимум1 := 1
  иначе
   |Проверяем, не равно ли очередное
   |значение
   |"старому" первому максимуму
   если a[i] = максимум1
    то
     |Встретился еще один первый максимум
     |Учитываем это
     кол_максимум1 := кол_максимум1 + 1
    иначе
     |Сравниваем ли очередное значение
     |со вторым максимумом
     если a[i] > максимум2
      то
       |Встретился элемент,
       |больший максимум2
       |Вторым максимумом станет
       |встреченный элемент
       максимум2 := a[i]
       |Пока он - единственный
       кол_максимум2 := 1
       |Значения максимум1
       |и кол_максимум1 не меняются
      иначе
       если a[i] = максимум2
        то
         |Встретился еще один
         |второй максимум
          кол_максимум2 :=
                        кол_максимум2 + 1
       все
     все
   все
 все
кц
```

4.9.2. Нахождение количества значений в массиве, равных элементу, который стоял бы на предпоследнем месте, если бы массив был отсортирован по неубыванию

При таком толковании понятия “второй максимум” задача несколько сложнее и требует учета двух важных обстоятельств.

Прежде всего ясно, что теперь значение *кол_максимум*1 всегда равно 1, а *кол_максимум*2 — может быть любым.

Кроме того, обращаем внимание на то, что на предпоследнем месте может стоять как элемент, равный максимальному, так и меньший его.

Для решения задачи, как и ранее, примем, что диапазон значений элементов массива известен, и после задания начальных значений *максимум*1 и *кол_максимум*1:

```
          максимум1 := -1001
          кол_максимум1 := 0
```

— рассматриваем все элементы, сравнивая их сначала со значением *максимум*1, а затем (при необходимости) — и со значением *максимум*2:

```
нц для i от 1 до n
 если a[i] > максимум1
  |Встретился элемент, больший максимум1
  то
   |Бывший первый максимум станет вторым
   максимум2 := максимум1
   |а значение кол_максимум2 станет равно
   |"старому" значению величины
   |кол_максимум1
   кол_максимум2 := кол_максимум1
   |Первым максимумом станет
   |встреченный элемент
   максимум1 := a[i]
   |Внимание - именно в таком порядке!
   |Новый первый максимум встретился
   |впервые,
   кол_максимум1 := 1
  иначе
   |Проверяем, не равно ли очередное
   |значение "старому" первому максимуму
   если a[i] = максимум1
    то
     |Встретился еще один первый максимум
      |Рассматриваем 2 случая
      если кол_максимум2 = 0
       то
          |Он - первый среди равных
          |максимум1
          |Принимаем его в качестве
          |максимум2
          максимум2 := a[i]
```

```
          |(пока единственного)
          кол_максимум2 := 1
        иначе  |кол_максимум2 > 0
          |Встретилась еще одна "копия"
          |значения максимум1
          |Увеличиваем значение кол_максимум2
          кол_максимум2 := кол_максимум2 + 1
          |Значение максимум2 не меняем
       все
      иначе
       |Сравниваем ли очередное значение
       |со вторым максимумом
       если a[i] > максимум2
        то
        |Встретился элемент, больший максимум2
        |Вторым максимумом станет встреченный
        |элемент
         максимум2 := a[i]
         |Пока он - единственный
         кол_максимум2 := 1
         |Значения максимум1 и кол_максимум1
         |не меняются
        иначе
         если a[i] = максимум2
          то
         |Встретился еще один второй максимум
            кол_максимум2 := кол_максимум2 + 1
         все
       все
    все
 все
кц
```

Дополнение

При решении задачи варианта 1 из [6] понадобится определение не только количества первого и второго максимумов баллов за экзамен, но и фамилий учеников, набравших “первый максимум” и “второй максимум”. Анализ последнего фрагмента показывает, что в качестве *максимум*1 и *максимум*2 в нем учитываются только первые встретившиеся значения из нескольких одинаковых. Прежде чем представлять фрагмент программы, в котором определяются также указанные фамилии (*фам_максимум*1 и *фам_максимум*2), заметим, что в нем обрабатывается не массив, а последовательность чисел.

```
максимум1 := -1001
кол_максимум1 := 0
фам_максимум1 := ""
нц для i от 1 до n
  …
 |Для i-го ученика его фамилия
 |и балл известны
 |(фамилия и балл)
 если  балл > максимум1
  |Встретился балл, больший максимум1
  то
   |Бывший первый максимум станет вторым,
   максимум2 := максимум1
   |фамилия - аналогично,
   фам_максимум2 := фам_максимум1
   |а значение кол_максимум2 станет равно
   |"старому" значению величины
   |кол_максимум1
   кол_максимум2 := кол_максимум1
   |Первым максимумом станет балл
   |текущего ученика
   максимум1 := балл
   |Запоминаем фамилию этого ученика
   фам_максимум1 := фамилия
   |Новый первый максимум встретился
   |впервые,
   кол_максимум1 := 1
 иначе
  |Проверяем, не равно ли значение балл
  |"старому" первому максимуму
  если балл = максимум1
   то  |Встретился еще один первый максимум
       |Рассматриваем 2 случая
     если кол_максимум2 = 0
      то
        |Он - первый среди равных максимум1
        |Принимаем его в качестве максимум2
        максимум2 := балл
        |(пока единственного)
        кол_максимум2 := 1
        |Запоминаем фамилию этого ученика
        фам_максимум2 := фамилия
      иначе  |кол_максимум2 > 0
        |Встретилась еще одна "копия"
        |значения максимум1
        |Увеличиваем значение кол_максимум2
        кол_максимум2 := кол_максимум2 + 1
        |Значения максимум2 и
        |фам_максимум2 не меняем
     все
    иначе
     |Сравниваем балл очередного ученика
     |со вторым максимумом
     если балл > максимум2
      то |Встретился балл, больший максимум2
       |Вторым максимумом станет этот балл
       максимум2 := a[i]
       |Запоминаем фамилию этого ученика
       фам_максимум2 := фамилия
       |Пока он - единственный
       кол_максимум2 := 1
       |Значения максимум1, кол_максимум1
       |и фам_максимум1 не меняются
      иначе
       если балл = максимум2
        то
       |Встретился еще один второй максимум
          кол_максимум2 := кол_максимум2 + 1
       |Значение фам_максимум2 не меняется
       все
     все
```

```
    все
  все
кц
```

4.10. Нахождение количества вторых минимумов

Все варианты этой задачи решаются аналогично предыдущей (конечно, с необходимыми изменениями).

4.11. Нахождение третьего максимума

"Третьим максимумом" будем называть элемент, который стоял бы на третьем справа месте, если бы весь массив был отсортирован по неубыванию.

Задача решается аналогично задаче 4.5 (в первом толковании термина):

```
максимум1 := ...
максимум2 := ...
|Значение максимум3 можно не задавать
нц для i от 1 до n
  если a[i] > максимум1
    |Встретился элемент, больший максимум1
    то
     |Бывший второй максимум станет третьим
     максимум3 := максимум2
     |Бывший первый максимум станет вторым
     максимум2 := максимум1
     |Первым максимумом станет встреченный
     |элемент
     максимум1 := a[i]
     |Внимание - именно в таком порядке!
    иначе
     |Очередной элемент не больше максимум1
     |Сравниваем его со значением максимум2
     если a[i] > максимум2
       |Встретился элемент, больший максимум2
       то
       |Бывший второй максимум станет третьим
        максимум3 := максимум2
        |а встреченный элемент принимаем его
        |в качестве
        |нового значения максимум2
        максимум2 := a[i]
        |Значение максимум1 не меняется
      иначе
       |Очередной элемент не больше максимум2
       |Сравниваем его со значением максимум3
       если a[i] > максимум3
        то
         |Меняем только значение максимум3
         максимум3 := a[i]
       все
    все
 все
кц
```

4.12. Нахождение третьего минимума

Задача решается аналогично предыдущей (конечно, с необходимыми изменениями).

Задания для самостоятельной работы учащихся

1. В массиве хранится информация о стоимости 1 килограмма 20 видов конфет. Определить, сколько стоят самые дешевые конфеты.

2. Известны расстояния от Москвы до нескольких городов. Найти расстояние от Москвы до самого удаленного от нее города из представленных в списке городов.

3. Известны результаты каждого из участников соревнований по лыжным гонкам (время, затраченное на прохождение дистанции гонки). Спортсмены стартовали по одному. Результаты даны в том порядке, в каком спортсмены стартовали. Определить, каким по порядку стартовал лыжник, показавший лучший результат. Если таких спортсменов несколько, то должен быть найден первый из них.

4. В массиве хранится информация о количестве осадков, выпавших за каждый день июля. Определить дату самого дождливого дня. Если таких дней было несколько, то должна быть найдена дата:

а) первого из них; б) последнего из них.

5. Известна информация о максимальной скорости каждой из 12 марок легковых автомобилей. Определить скорость автомобиля, больше которой только максимальное значение в массиве.

6. В массиве хранится информация о результатах 22 спортсменов, участвовавших в соревнованиях по бегу на 100 м. Определить результаты спортсменов, занявших первое и второе места. Задачу решить, не используя два прохода по массиву.

7. В одном массиве записаны названия 20 команд — участниц чемпионата по футболу, в другом — соответствующее им количество набранных очков. Определить команды, занявшие первое и второе места. Задачу решить, не используя два прохода по массиву.

8. Известна информация о среднедневной температуре за каждый день июля. Определить даты двух самых холодных дней.

9. Известна информация о баллах, набранных каждым из 25 учеников на ЕГЭ по информатике. Определить, сколько учеников набрали максимальную сумму баллов.

10. В массиве записана информация о весе в кг каждого из 30 человек. Вывести на экран:

1) в первой строке — порядковые номера людей, имеющих максимальный вес среди представленных;

2) во второй строке — их количество.

5. Разные задачи

5.1. Суммирование значений для различных категорий

Анализ решения

Задача этого типа в общем виде формулируется так: "Дана последовательность чисел, каждое из

которых относится к некоторой категории (номеру школы, номеру параллели класса и т.п.). Необходимо определить сумму всех чисел для каждой категории".

Можно также сформулировать задачу по-другому: "Даны *N* пар чисел, второе из которых в каждой паре определяет некоторую категорию (номер школы, номер параллели класса и т.п.). Необходимо для каждой категории определить сумму чисел, задаваемых первыми в паре".

Как и при решении задач группы 2, целесообразно использовать массив с индексами, соответствующими значениям категорий (возможно, и неиспользуемыми). В этот массив и будем записывать искомые значения. Если его имя — *всего*, то фрагмент, в котором решается задача, имеет вид:

```
|Обнуляем элементы массива всего
нц для i от … до …
|Используются границы диапазона категорий
 всего[i] := 0
кц
|Вводим и учитываем числа
нц для i от 1 до N
 ввод число, категория
 |Учитываем значение число в массиве всего
 |для соответствующей категории
 всего[категория] := всего[категория] +
                     число
кц
```

Язык Паскаль

```
{Обнуляем элементы массива всего}
for i := … to … do
{Используются границы диапазона категорий}
  vsego[i] := 0;
for i := 1 to N do
  begin
   readln(chislo, kategoria);
   vsego[kategoria] := vsego[kategoria] +
                       chislo
  end
```

Если суммируемые числа являются частью строки, то их необходимо предварительно выделить (см. раздел 1).

5.2. Расчет среднего значения с точностью до целых

Для расчета среднего арифметического нескольких чисел с точностью до целых необходимо определить сумму всех чисел (*сум*) и их количество (*кол*), а затем рассчитать искомое значение *сред*:

```
сред := int(сум/кол)
```

Если обрабатываемые числа — целые, то задача может быть решена также следующим образом:

```
сред := div(сум, кол)
```

— где *div* — функция, возвращающая целую часть от деления первого параметра на второй.

Язык Паскаль

```
sred := trunc(sum/kol);
```

или

```
sred := sum div kol;
```

5.3. Преобразование строкового представления числа в число

В школьном алгоритмическом языке преобразование строкового представления целого или вещественного числа в число выполняется, соответственно, с помощью стандартных функций *лит_в_цел* и *лит_в_вещ*, общий вид которых:

```
лит_в_цел(строка, успех)
```

и

```
лит_в_вещ(строка, успех)
```

— где *строка* — преобразуемая строка, *успех* — величина логического типа. Если *строка* содержит только целое/вещественное число, то величине *успех* присваивается значение `да`, и функция возвращает соответствующее число, в противном случае величине *успех* присваивается значение `нет`, и функция возвращает 0.

В языке Паскаль для решения указанной задачи используется процедура `val`. Кроме того, в случае преобразования символов-цифр может быть применена функция `ord`.

Задания для самостоятельной работы учащихся

1. После единых выпускных экзаменов по информатике в район пришла информация о том, какой ученик какой школы сколько набрал баллов. Необходимо определить сумму баллов, набранных всеми учениками каждой школы. Экзамен сдавали ученики школ с номерами от 1-го до 20-го.

2. Для условий предыдущей задачи для каждой школы определить средний балл ее учеников с точностью до целых.

3. Дана строка формата:

<двузначное число> <пробел> <цифра>
<пробел> <трехзначное число>

Определить сумму трех числовых значений.

В заключение в качестве примера приведем разбор задачи С4 из варианта 1 задания в книге [5].

Условие[3]

На вход программе подаются сведения о номерах школ учащихся, участвовавших в олимпиаде. В первой строке сообщается количество учащихся *N*, каждая из следующих *N* строк имеет формат:

<Фамилия> <Инициалы> <номер школы>,

где <Фамилия> — строка, состоящая не более чем из 20 символов, <Инициалы> — строка, состоящая из четырех символов (буква, точка, буква, точка), <номер школы> — не более чем двузначный номер. <Фамилия> и <Инициалы>, а также

[3] Условие цитируется согласно соответствующему источнику.

<Инициалы> и <номер школы> разделены одним пробелом. Пример входной строки:

Иванов П.С. 57

Требуется написать как можно более эффективную программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет выводить на экран информацию, из какой школы было больше всего участников (таких школ может быть несколько). Также программа должна подсчитать общее количество школ, приславших больше всего участников.

Следует учитывать, что $N >= 1000$.

Анализ решения

Ясно, что необходимо узнать количество участников олимпиады от каждой школы. Для хранения информации об этом применим массив с именем *всего* с индексами от 1 до 99. Заполнение этого массива следует провести во время ввода исходных данных после выделения номера школы, в которой учится очередной участник.

Методика выделения числа после двух слов рассмотрена в пункте 1.6, а задача заполнения массива *всего* — это аналог типовой задачи 2.5.

Можно также сразу выделить номер школы (по его "местонахождению" в строке), не пропуская фамилию и инициалы ученика:

```
если строка[длин(строка) - 1] = " "
 то |Номер школы - однозначный
 школа := лит_в_цел(строка[длин(строка)],
                    успех)
 иначе |Двузначный
  школа := лит_в_цел(строка[длин(строка) - 1
           : длин(строка)], успех)
все
```

Далее, если бы требовалось подсчитать общее количество школ, приславших больше всего участников, то это можно было бы сделать за один проход по массиву *всего* (см. типовую задачу 4.7). Но поскольку нужна также информация о том, из какой школы было больше всего участников (а в ходе одного прохода соответствующие данные изменяются), то значения искомых величин надо определять за два прохода:

```
…
|1. Ищем максимальный элемент массива всего
|(см. типовую задачу 4.1)
макс := всего[1]
нц для i от 2 до 99
  если всего[i] > макс
   то
    макс := всего[i]
  все
кц
|2. Выводим номера школ с максимальным
|числом участников,
|одновременно подсчитывая их количество k,
|(см. типовые задачи 3.3 и 3.1)
  k := 0
нц для i от 1 до 99
  если всего[i] = макс
   то
    вывод i, " "
    k := k + 1
  все
кц
|которое затем выводим на экран
вывод нс, "Количество школ, приславших
наибольшее число участников ", k
кон
```

Итак, общая структура программы решения обсуждаемой задачи такая:

```
  1. Заполнение массива всего нулевыми
значениями
  2. Ввод значения N
  3. Цикл для i от 1 до N |Ввод и обработка
входных строк
    3.1. Ввод информации об i-м ученике
    3.2. Выделение номера школы (ТЗ 1.6[4])
    3.3. Учет этой школы в массиве всего
(ТЗ 2.5)
  конец цикла
  4. Определение максимального элемента
массива всего (ТЗ 4.1)
  5. Поиск и вывод необходимых номеров
школ (с подсчетом значений k)
  6. Вывод значения k (см. выше)
```

Полностью программа на школьном алгоритмическом языке (как и программы решения других задач) будет представлена на диске к данному номеру "Информатики".

Литература

1. Демонстрационный вариант Единого государственного экзамена по информатике и ИКТ 2010 года. *http://www.fipi.ru/binaries/895/inf.zip*

2. Демонстрационный вариант Единого государственного экзамена по информатике и ИКТ 2009 года. / *http://www.fipi.ru/binaries/731/infZIP%20-%20WinRAR.zip*

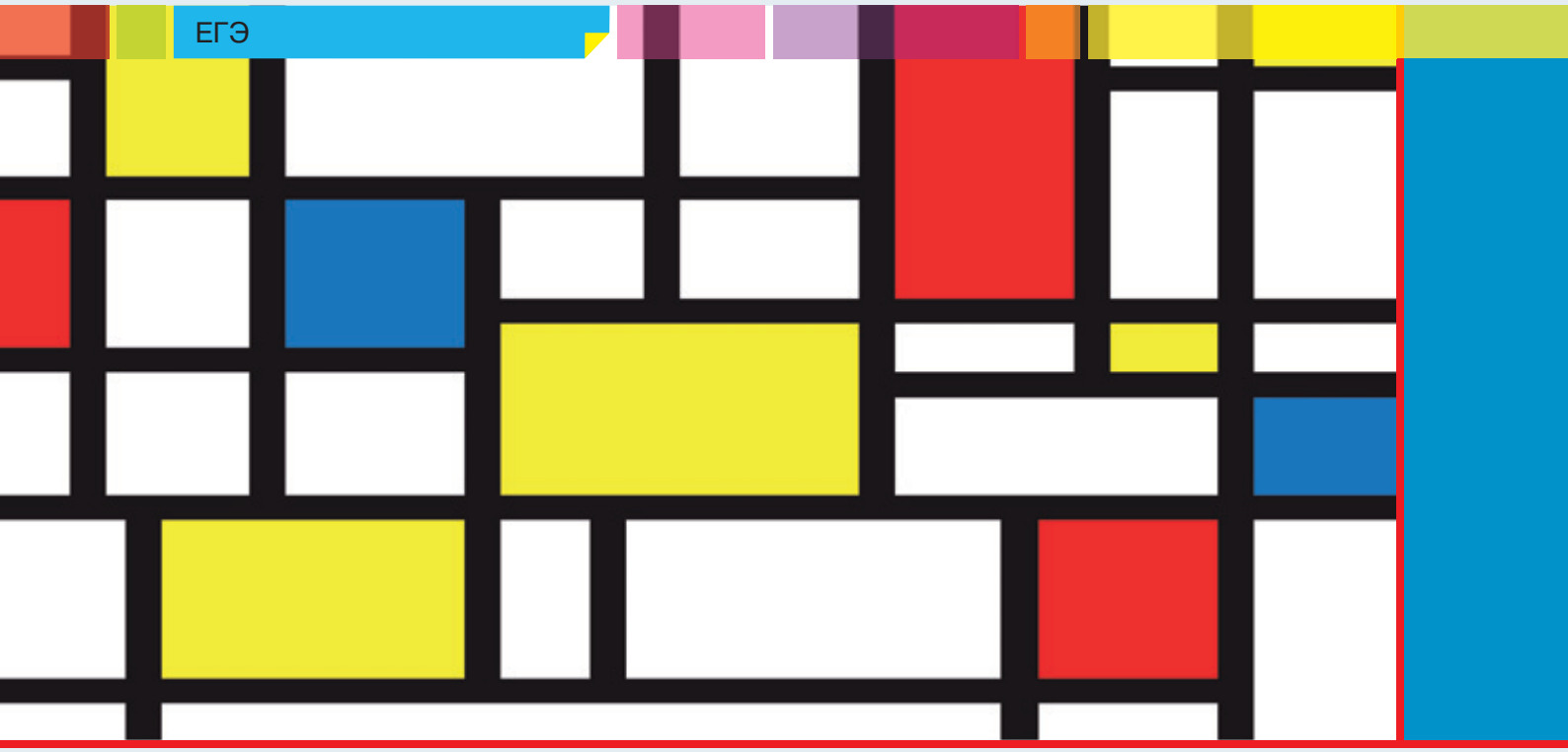
3. Демонстрационный вариант Единого государственного экзамена по информатике и ИКТ 2008 года. */http://www.fipi.ru/binaries/518/inform.rar*

4. Демонстрационный вариант Единого государственного экзамена по информатике и ИКТ 2007 года. / *http://www.fipi.ru/binaries/395/inf%20dem.doc*

5. *Якушкин П.А., Лещинер В.Р., Кириенко Д.П.* ЕГЭ-2010. Информатика. Типовые тестовые задания. М.: изд-во "Экзамен", 2010.

6. Самое полное издание типовых вариантов реальных заданий ЕГЭ-2010: Информатика. / Авт.-сост. *П.А. Якушкин, Д.М. Ушаков*. М.: Астрель, 2010 (Федеральный институт педагогических измерений).

[4] Здесь и ниже ТЗ обозначает ссылку на типовую задачу и ее номер.



Задачи “на интервалы”: наглядно-графический метод решения

О.Б. Богомолова, д. п. н., учитель информатики и математики ГОУ СОШ № 1360 г. Москвы

Д.Ю. Усенков, ст. н. с. Института информатизации образования Российской академии образования, Москва

Среди задач, предлагаемых на Едином государственном экзамене по информатике, есть достаточно большая группа заданий, связанных с логическими выражениями:

1) задания на преобразование логических выражений с использованием законов алгебры логики — требуется определить, какому из приведенных в вариантах ответа выражению эквивалентно заданное логическое выражение;

2) задания на построение таблицы истинности — требуется определить, какое логическое выражение соответствует заданному фрагменту таблицы истинности, либо составить таблицу истинности для логической функции с параметрами в виде десятичных чисел;

3) задания на составление логического выражения, включающего в себя запись операций сравнения — “задачи с гласными и согласными буквами в именах”;

4) задания на определение значения параметра (переменной) — требуется определить значение или диапазон значений входящей в запись логического выражения переменной (точнее — в запись входящих в его состав операций сравнения), при которых данное выражение ложно или истинно;

5) задания на определение количества возможных решений логического уравнения либо на указание самих этих решений как комбинаций входящих в запись уравнения нескольких переменных;

6) появившиеся в ЕГЭ 2011 года задания на определение количества возможных решений системы логических уравнений.

Авторы этой статьи ранее уже предлагали читателям разбор некоторых подобных задач (см. статьи “Логические задачи на ЕГЭ: имена и логические выражения” — № 8 за 2011 г. и “По следам ЕГЭ-2011: новые задачи” — № 15 за 2011 г.). Теперь же мы предлагаем вернуться к этой теме.

На сайте К.Полякова (*http://kpolyakov.narod.ru/school/ege.htm/pshop.htm*) представлен целый ряд задач ЕГЭ, в том числе упомянутые логические (раздел “Логика”, файл “B10 — преобразование логических выражений”), с разбором их решения. Однако при попытке ис-

пользования этих материалов в ходе подготовки учащихся к ЕГЭ выяснилось, что предлагаемые способы решения, представленные в форме рассуждений, дети осваивают далеко не всегда. Впрочем, абстрактные, "текстовые" описания во многих случаях можно заменить гораздо более наглядным графическим решением.

Для начала (поскольку "повторение — мать учения" ☺) рассмотрим задачу, которую "традиционно" принято решать графическим способом.

Задача 1 (демонстрационный вариант 2009 г., задание В4). Каково наибольшее целое число X, при котором истинно высказывание

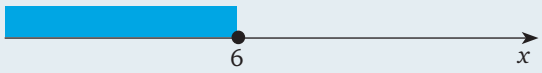
$$(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))?$$

Решение[1]

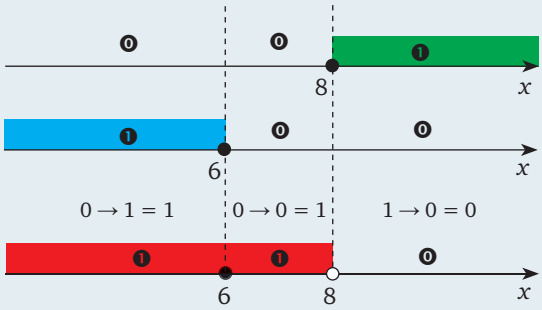
1. Интервал истинности (в целых числах) для условия $50 < X \cdot X$:

2. Интервал истинности для условия

$$50 > (X + 1) \cdot (X + 1):$$



3. Операция следования (→) для этих интервалов. Выделенные значения соответствуют логической единице, а невыделенные — нулю. Операция следования дает значение 0, когда из 1 следует 0, а в остальных случаях получается значение 1.



Граничная точка 8 не входит в результирующий "единичный" интервал. Следовательно, наибольшее целое значение X, удовлетворяющее условию задачи, равно 7.

А можно ли решить эту задачу проще?

Сразу обратим внимание на то, что операция следования дает результат "Истина" в трех случаях из четырех возможных, а результат "Ложь" — только в одном случае из четырех, когда из 1 ("Истина") следует 0 ("Ложь"). Поэтому нам было бы удобнее решать предложенную задачу в "обратной" формулировке:

Задача 1*а*. Каково *наименьшее* целое число, при котором *ложно* высказывание

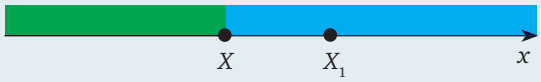
$$(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))?$$

[1] Более подробно см. в статье "Логические задачи на ЕГЭ: имена и логические выражения" в № 8 за 2011 г.

Решение

Прежде всего докажем правомерность такой замены формулировки задачи.

Предположим, что искомое значение X в первоначальной задаче найдено. Раз в исходной задаче оно названо наибольшим целым числом, то это означает, что заданное выражение истинно на диапазоне значений $(-\infty, X]$. Тогда на оставшейся части числовой оси, т.е. в диапазоне $(X, +\infty)$, данное выражение будет ложно. А значит, определив наименьшее целое значение X_1, при котором заданное выражение ложно, мы легко получим по нему искомое наибольшее X, при котором это выражение истинно: $X = X_1 - 1$.



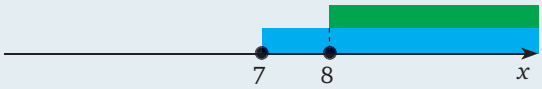
Решим задачу по предложенному новому способу.

1) Выражение $(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))$ ложно, когда

$$\begin{cases} 50 < X \cdot X - \textit{истинно}; \\ 50 > (X + 1) \cdot (X + 1) \ - \textit{ложно}. \end{cases}$$

2) Первой операции сравнения соответствует диапазон целых чисел $[8, +\infty)$. Второй операции сравнения (учитывая, что она должна быть ложной, т.е. истинно сравнение $50 \le (X + 1) \cdot (X + 1)$) соответствует диапазон целых чисел $[7, +\infty)$.

3) Графическое представление решения "модифицированной" задачи.



4) То, что оба условия сравнения объединены в систему, означает, что оба они должны выполняться одновременно. Следовательно, решением этой системы уравнений является *пересечение* построенных интервалов (пересечение множества составляющих их целых чисел). Этот интервал ложности операции следования — $[8, +\infty)$.

5) Наименьшее целое значение (наше X_1), при котором заданное выражение ложно, равно 8.

А теперь вспомним, что искомое значение X в первоначальной задаче (наибольшее целое число, при котором исходное логическое выражение истинно) на 1 меньше найденного нами значения X_1. Следовательно, для исходной задачи ответ — число 7.

А теперь рассмотрим другие задачи, которые, по сути, сводятся к ранее рассмотренной. Задания взяты с сайта К.Полякова (*http://kpolyakov.narod.ru/school/ege.htm/pshop.htm*, файл "B10 — преобразование логических выражений"), где приводится разбор их решений путем рассуждений. Это позволит читателям сравнить эти решения с предлагаемым здесь графическим способом.

Задача 2. A, B и C — целые числа, для которых истинно высказывание:

$$\neg(A = B) \wedge ((A > B) \rightarrow (B > C)) \wedge ((B > A) \rightarrow (C > B))$$

Чему равно B, если $A = 45$ и $C = 43$?

Решение

Вместо того чтобы сразу пытаться преобразовывать исходное выражение или пытаться интуитивно определить предполагаемый правильный ответ, просто подставим заданные значения переменных A и C в исходное выражение:

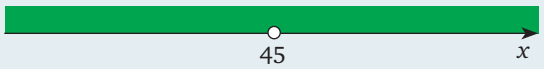
$$\neg(45 = B) \wedge ((45 > B) \rightarrow (B > 43)) \wedge$$
$$\wedge ((B > 45) \rightarrow (43 > B))$$

Теперь аналогия этой задачи с ранее рассмотренной (найти наименьшее или наибольшее целое значение, при котором истинно заданное выражение) становится очевидной. Соответственно, аналогичным может быть и ее решение при помощи интервалов.

1) Замечаем, что выражение состоит из трех компонентов, соединенных операцией “И”. Значит, чтобы это выражение было истинным, нужно обеспечить истинность всех трех этих компонентов:

$$\begin{cases} \neg(45 = B) - \textit{истинно}; \\ (45 > B) \rightarrow (B > 43) \ - \textit{истинно}; \\ (B > 45) \rightarrow (43 > B) - \textit{истинно}. \end{cases}$$

2) Для первого выражения этой системы получаем интервал истинности (в целых числах): $(-\infty, 45) \cup (45, +\infty)$ (такая запись с объединением двух интервалов фактически означает всю числовую прямую, кроме числа 45).

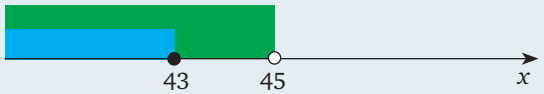


3) Разберем второй компонент системы: $(45 > B) \rightarrow (B > 43)$. Он будет истинным в трех случаях из четырех, поэтому нам проще будет решать “обратную” задачу — найти, при каких значениях B это выражение ложно (один случай из четырех), а потом взять значения на числовой прямой, которые *не входят* в найденный интервал ложности этого выражения.

Указанная операция следования ложна, если

$$\begin{cases} 45 > B - \textit{истинно}; \\ B > 43 \ - \textit{ложно}. \end{cases}$$

Первой части соответствует интервал истинности $(-\infty, 45)$, а второй — интервал ложности $(-\infty, 43]$. Тогда интервал ложности рассматриваемой операции следования: $(-\infty, 43]$.



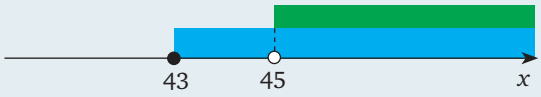
Отсюда интересующий нас интервал истинности второго компонента исходного выражения: $(43, +\infty)$.

4) Аналогично разберем третий компонент системы: $(B > 45) \rightarrow (43 > B)$, также применив прием замены поиска интервала истинности выражения поиском интервала его ложности.

Указанная операция следования ложна, если

$$\begin{cases} B > 45 - \textit{истинно}; \\ 43 > B \ - \textit{ложно}. \end{cases}$$

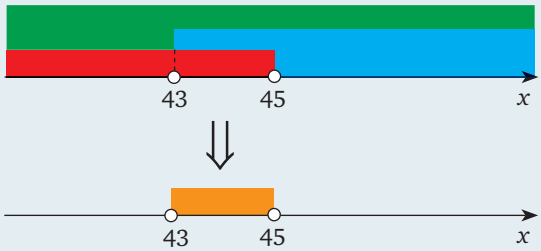
Первой части соответствует интервал истинности $(45, +\infty)$, а второй — интервал ложности $[43, +\infty)$. Тогда интервал ложности рассматриваемой операции следования: $(45, +\infty)$.



Отсюда интересующий нас интервал истинности третьего компонента исходного выражения: $(-\infty, 45]$.

5) А теперь возвращаемся к системе этих трех компонентов и ищем пересечение полученных для них интервалов истинности:

$$((-\infty, 45) \cup (45, +\infty)) \cap (43, +\infty) \cap (-\infty, 45] = (43, 45).$$



Как видим, интервал истинности исходного выражения — от 43 до 45, *не включая эти граничные точки*. Следовательно, единственное возможное целочисленное решение этой задачи — число 44.

Итак, графическое решение получилось несколько более длинным, чем решение путем логических рассуждений. Однако реально оно выполняется быстрее, поскольку здесь все операции выполняются “механически” (надо только быть внимательным при обмене местами левой и правой частей в записи неравенств и при переходе от истинности к ложности операций сравнения), и более понятно учащимся благодаря его наглядности. Кроме того, в данном конкретном случае “интуитивное” решение, приведенное у К.Полякова, возможно за счет особенности выражения, заданного в условии задачи, тогда как предложенный выше графический способ универсален и приводит нас к решению для любого исходного выражения.

А теперь для закрепления пройденного рассмотрим графическое решение еще одной задачи подобного типа.

Задача 3[2]. Сколько существует целых значений X, при которых ложно высказывание:

$$(|X| \geq 5) \vee (|X| < 1)?$$

Решение

1) Два компонента данного выражения связаны операцией “ИЛИ”, которая может быть ложной в

[2] Задача № 47 для самостоятельного решения, сайт *http://kpolyakov.narod.ru/school/ege.htm/pshop.htm*, файл “B10 — преобразование логических выражений”.

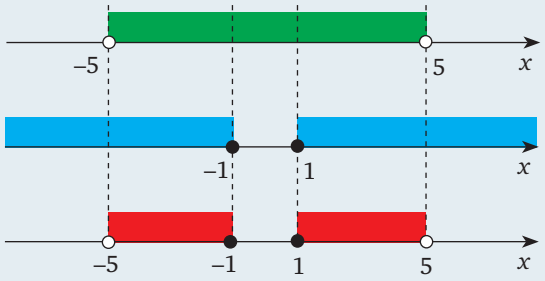
одном случае из четырех возможных — когда оба эти компонента ложны. Тогда это выражение эквивалентно системе:

$$\begin{cases} |X| \geq 5 - \textit{ложно}; \\ |X| < 1 - \textit{ложно}. \end{cases}$$

2) Рассмотрим первый компонент. Очевидно (если вспомнить понятие модуля), его интервал истинности имеет вид: $(-\infty, -5] \cup [5, +\infty)$, а интервал ложности составляет всю оставшуюся часть числовой оси: $(-5, 5)$.

3) Рассмотрим второй компонент. Аналогично, его интервал истинности имеет вид: $(-1, 1)$, тогда интервал ложности составляет: $(-\infty, -1] \cup [1, +\infty)$.

4) Строим пересечение полученных ранее интервалов для первого и второго компонентов:



5) В пределах полученного составного интервала ($(-5, -1] \cup [1, 5)$), учитывая, что краевые значения –5 и 5 в него не входят, расположены следующие целые числа: –4, –3, –2, –1, 1, 2, 3, 4. Всего — 8 значений переменной X.

Завершая статью, еще раз повторим правила, которыми нужно руководствоваться при решении логических задач с интервалами:

1) при обмене местами левой и правой частей неравенства его знак меняется на противоположный;

2) если неравенство является ложным, то эквивалентное ему истинное неравенство не только имеет противоположный знак, но и становится из строгого нестрогим и наоборот; то же самое происходит при замене истинного неравенства эквивалентным ему ложным;

3) соединение компонентов логического выражения операцией "И" соответствует *пересечению* интервалов их истинности (интервалов значений, при которых эти компоненты истинны); соединение компонентов логического выражения операцией "ИЛИ" соответствует *объединению* интервалов их истинности;

4) интервал ложности представляет собой всю часть числовой прямой, кроме интервала истинности этого выражения — производится *вычитание* интервала истинности из числовой прямой; аналогично определяется и интервал истинности по интервалу ложности.

Кроме того, при решении задач с интервалами надо внимательно читать текст их условия: если в вопросе фигурирует, например, "наибольшее натуральное число X" или "наибольшее целое положительное число X", то это означает добавление дополнительного условия — $X > 0$.

"Программиста бьют по рукам, если он посмеет написать оператор цикла, не найдя перед этим его инварианта"

В теме "Теория алгоритмов" многим нашим коллегам не нравится первое слово — "теория". Действительно, "повествовательные" лекционные темы все менее подходят для школьного курса информатики, даже профильного. Но… В общем, хотя у каждого из нас среди тем обязательно есть "любимчики", практически о любой составляющей курса можно сказать, что если нам не нравится, то скорее всего… ☺

В следующем номере мы познакомимся с главой "Элементы теории алгоритмов" из нового разрабатываемого учебника для профильного курса (ранее мы уже публиковали некоторые разделы учебника, в частности, совсем недавно — главу "Основы объектно-ориентированного программирования").

Глава "Элементы теории алгоритмов" будет поддержана программным обеспечением — интерпретаторами машин Поста, Тьюринга и реализацией нормальных алгорифмов (это не ошибка Маркова). Указанные темы можно отнести к достаточно традиционным. Менее традиционные (точнее — немного подзабытые) разделы главы посвящены обсуждению вопросов, связанных со сложностью алгоритмов и доказательным программированием. Все хорошо помнят лозунг академика А.П. Ершова про то, что "программирование — вторая грамотность", но не все знают другое его утверждение — "программиста бьют по рукам, если он посмеет написать оператор цикла, не найдя перед этим его инварианта".

Понятие инварианта — одно из основных, обсуждаемых в разделе, посвященном доказательному программированию.

По следам новой теории ввода-вывода

К.Ю. Поляков, д. т. н., учитель информатики ГОУ СОШ № 163, Санкт-Петербург

В статье [1] А.А. Дуванов предлагает свое толкование привычных, казалось бы, терминов “устройство ввода” и “устройство вывода”. По версии [1], устройства ввода — это любые устройства, через которые компьютер получает данные, в том числе дисководы, модемы, флэш-диски и пр. Аналогично устройства вывода — это любые устройства, через которое данные выводятся из компьютера, причем получателем может быть как человек, так и другое цифровое устройство (те же дисковод и флэш-диск).

Как известно, об определениях не спорят. Напротив, спор в научном мире начинается с фиксации определений. Иногда он на этом и заканчивается, поскольку спор умных людей нередко вызван именно тем, что его участники называют одним и тем же термином совершенно разные вещи. Поэтому некорректно говорить о том, верные определения даны в [1] или неверные. Но можно сравнить их с альтернативными и проследить, к чему они в конечном счете приводят: какую картину миру удается построить на основе этих определений.

Начнем с официальных источников. Согласно ГОСТу 25868-91 [2], “**устройство ввода (вычислительноймашины)**—периферийное устройство, обеспечивающее преобразование информации в форму, необходимую для ее автоматического ввода в ЭВМ”, и “**устройство вывода (вычислительной машины)** — периферийное устройство, обеспечивающее вывод данных из ЭВМ”. Приведенное в ГОСТе определение устройства ввода явно подчеркивает его характерную черту — первичное **преобразование информации** из “некомпьютерного” вида в “компьютерный”. Таким образом, определение из [1] противоречит определению в ГОСТе, что само по себе уже нехорошо, потому что способно только вызвать путаницу. Стандарт он тем и хорош, что создает общепринятый фундамент для дальнейших “строительных” работ.

Определение устройства вывода в ГОСТе значительно слабее, но по аналогии, сохраняя общую идеологию, хочется написать так: “**устройство вывода (вычислительной машины)** — периферийное устройство, обеспечивающее преобразование данных в форму, доступную для ее восприятия человеком”.

В результате можно сделать следующий вывод: определения в [1] не позволяют как-то выделить (классифицировать) устройства, которые выполняют преобразование данных из “некомпьютерного” формата в “компьютерный” и обратно, то есть находятся на границе между “некомпьютерным” и “компьютерным” мирами. В сравнении с определениями ГОСТа

это шаг назад, поскольку среди устройств ввода/вывода (по терминологии [1]) затерялся целый подкласс устройств, обладающих *принципиальными* отличиями от всех остальных.

Теперь представим на минуту, что куда-то исчезли все устройства, которые по ГОСТу назывались устройствами ввода. Вместе с ними исчезли и устройства вывода — мониторы, принтеры, наушники и т.д. Остались только флэшки, дисководы, роутеры, коммутаторы, модемы и пр. Что получим? "Компьютерный" мир оказывается замкнут, человек теряет все средства общения с ним, поскольку воспринимать цифровые сигналы непосредственно мы пока не научились. Этот факт говорит о том, что исходные определения упустили что-то важное и устройства ввода (вывода) бывают разные.

Теперь проанализируем выводы, к которым приходит автор [1] на основе введенных им определений.

"Итак, компьютер — это процессор и внутренняя память".

"Все другие устройства обеспечивают или ввод информации в память компьютера, или вывод из нее. Соответственно, все другие устройства, подключаемые к процессору и памяти, являются устройствами ввода и(или) вывода".

Вряд ли кто-то будет спорить с тем, что компьютер — это система, обладающая системным свойством — обрабатывать данные по введенной в него программе. Если мы говорим, что "*компьютер — это процессор и внутренняя память*", возникает вопрос о том, обладает ли эта пара тем же самым системным свойством? На наш взгляд, нет, потому что ввести в него программу не удастся, так же как и получить какие-то результаты.

"Флэшка и принтер между собой принципиально не отличаются".

По мнению автора данной заметки, это неверное утверждение. Принтер выполняет преобразование информации из цифровой формы в "бумажную". Флэш-диск — это устройство "компьютерного" мира, которое никак не обменивается данными с реальным миром. Смешение двух понятий похоже на ситуацию в индейском племени хипо, в языке которого зеленый и голубой цвета обозначались одним и тем же словом, и сначала предполагалось, что индейцы не различают эти цвета вообще.

"На схеме зеленые стрелки показывают движение информации, а красные — управляющие воздействия процессора":

При анализе этой схемы возникает несколько серьезных вопросов. Возьмем, для примера, флэш-диск, который часто используется для иллюстрации в [1]. Согласно предлагаемой терминологии, его можно считать устройством ввода и вывода, то есть внешним устройством (левый блок на схеме). В таком диске есть контроллер и память. Если с памятью все понятно, то на месте контроллера на схеме написано "устройство ввода/вывода". Кроме того, судя по схеме, процессор только управляет прямым доступом к памяти и не может обмениваться данными с внутренней памятью и напрямую с внешним устройством (нет зеленых стрелок), что, мягко говоря, не соответствует истине.

При чтении статьи [1] в воздухе постоянно витает вопрос "зачем?". Новые определения понятий "устройство ввода" и "устройство вывода" не имеют, на наш взгляд, никаких преимуществ перед классическими, закрепленными в ГОСТе. Картина немного проясняется в последних разделах статьи [1]: "Теория относительности ввода и вывода" и "Формальные определения", где утверждается, что

• понятия "устройство ввода" и "устройства вывода" применимы не только к компьютерам, но и к любому процессу передачи информации вообще:

"Так что же есть сканер? Для компьютера это устройство ввода.

А для человека? А для человека сканер — устройство вывода. Мы ведь отдаем ему листок, "выводя" из папки, в которой листок хранили";

• эти понятия зависят от направления передачи информации; два компьютера, обменивающиеся данными, попеременно являются друг для друга устройствами ввода и вывода.

Получается такая кибернетическая картина мира, в которой человеку отводится примерно такая же роль, как и любому компьютерному устройству. С точки зрения [1], компьютер и человек могут рассматриваться как устройства ввода и вывода для "флэшки".

"Ну и что?" — скажет читатель. Да просто предложен еще один философско-кибернетический взгляд на мир. Более важный вопрос: "Что из него следует?" Да в общем-то ничего и не следует. "Тогда зачем?" Этот вопрос пока без ответа.

Подведем итоги. По мнению автора этой заметки, понятия "устройства ввода" и "устройства вывода" должны быть на своем месте, обозначая устройства для преобразования данных между компьютерным и реальным мирами. Попытки расширить эти понятия пока ни к чему не ведут.

Литература

1. *А.А. Дуванов.* Устройства ввода и вывода. Теория относительности // Информатика, № 14, 2011, с. 24–29.

2. ГОСТ 25868-91. Периферийное оборудование систем обработки информации. Термины и определения.

Общероссийский проект «Школа цифрового века» по комплексному обеспечению образовательных учреждений методической интернет-поддержкой разработан в соответствии с программой модернизации системы общего образования России и направлен на повышение профессионального уровня педагогических работников

Общероссийский проект

Школа цифрового века

Интернет-сопровождение проекта – Издательский дом «ПЕРВОЕ СЕНТЯБРЯ»

Прием заявок продолжается!

С 1 января 2012 года каждое образовательное учреждение, участвующее в проекте «Школа цифрового века», получает неограниченный доступ к электронным предметно-методическим журналам Издательского дома «Первое сентября».

Участвуйте в проекте всей школой!

Свежие номера журнала «ИНФОРМАТИКА» будут приходить в Ваш Личный кабинет на сайте www.1september.ru бесплатно!

Станьте обладателем сертификата «Учитель цифрового века»!


№0002977861

Общероссийский проект
Школа цифрового века

Интернет-сопровождение проекта – Издательский дом «ПЕРВОЕ СЕНТЯБРЯ»

Сертификат

«ШКОЛА ЦИФРОВОГО ВЕКА»

удостоверяет, что

_____

_____

_____

принял участие в Общероссийском проекте «Школа цифрового века», разработанном в соответствии с программой модернизации системы общего образования и направленным на комплексное Интернет-обеспечение образовательных учреждений цифровыми предметно-методическими материалами. Педагогический коллектив данного образовательного учреждения профессионально использует в своей работе возможности современных информационных технологий и цифровых образовательных ресурсов.

Председатель оргкомитета Общероссийского проекта «Школа цифрового века»

Подпись

МОСКВА - 2012


№0002977861

Общероссийский проект
Школа цифрового века

Интернет-сопровождение проекта – Издательский дом «ПЕРВОЕ СЕНТЯБРЯ»

Сертификат

«УЧИТЕЛЬ ЦИФРОВОГО ВЕКА»

удостоверяет, что

_____

_____

_____

применяет в работе современные информационные технологии и цифровые образовательные ресурсы. Эффективно использует возможности методической интернет-поддержки образовательного процесса, предоставленной в рамках Общероссийского проекта «Школа цифрового века».

Председатель оргкомитета Общероссийского проекта «Школа цифрового века»

Подпись

МОСКВА - 2012

Для образовательных учреждений, участвующих в проекте «Школа цифрового века» с 1 января по 30 июня 2012 года, оргвзнос – 2 тысячи рублей.

Коды доступа по числу педагогических работников предоставляются бесплатно.

Подробности на сайте
digital.1september.ru

ПОДПИСКА–2012

журнал

Информатика – Первое сентября

ТАРИФНЫЕ ПЛАНЫ НА ПОДПИСКУ
1-е полугодие 2012 г.

Максимальный – от 999 руб.
бумажная версия + CD + доступ к электронной версии на сайте

Подписаться можно на почте по каталогам «Роспечать» (индекс 32291), «Почта России» (индекс 79066) или на сайте www.1september.ru

Оптимальный – 594 руб.
электронная версия на CD + доступ к электронной версии на сайте

Подписаться можно на почте по каталогам «Роспечать» (индекс 19179), «Почта России» (индекс 12684) или на сайте www.1september.ru

Экономичный – 200 руб. **АКЦИЯ–2012***
доступ к электронной версии на сайте

Подписаться по данному тарифному плану можно только на сайте www.1september.ru

Бесплатный – 0 руб. **ШКОЛА ЦИФРОВОГО ВЕКА****
для педагогических работников образовательных учреждений, участвующих в Общероссийском проекте «Школа цифрового века». Подробности – на digital.1september.ru

Бумажная версия

CD с электронной версией журнала и дополнительными материалами для практической работы

Доступ к электронной версии журнала на сайте. Дополнительные материалы включены

Именные сертификаты – пользователям электронной версии на сайте www.1september.ru

ЭКОНОМИЧНЫЙ тарифный план

ОПТИМАЛЬНЫЙ тарифный план

МАКСИМАЛЬНЫЙ тарифный план

На сайте www.1september.ru подписку можно оплатить по кредитным картам VISA MasterCard

* **АКЦИЯ–2012**: подробности на с. 4 обложки этого номера.
** **ШКОЛА ЦИФРОВОГО ВЕКА**: подробности на с. 30 этого номера.

“О знаках и строках замолвите слово...”, или Несколько занимательных задач на работу со строками и множествами

Д.Ю. Усенков, ст. н. с. Института информатизации образования РАО, Москва

О.Б. Богомолова, д. п. н., учитель информатики и математики ГОУ СОШ № 1360, г. Москвы

При изучении в школе основ программирования по какой-то неизвестной авторам данной статьи причине работа с множественными типами данных (кроме разве что массивов и в некоторых случаях — файлов) почти не затрагивается. По крайней мере найти в различных публикациях или на сайтах образовательной тематики какие-либо хорошие (интересные и вместе с тем поучительные) задачи оказалось довольно сложно.

Желая “сломать сложившиеся стереотипы”, предлагаем читателям несколько подобных задач. Их условия в основном взяты из книги А.И. Гусева “Учимся информатике: задачи и методы их решения” (изд-во “Диалог-МИФИ”, 2004 г.), посвященной изучению программирования на языке Бейсик, однако мы разберем решение этих задач на более современном языке Паскаль (сопровождая их, правда, некоторыми комментариями, касающимися решения таких задач на Бейсике).

Наши цели при этом:

• во-первых, показать читателям (учащимся, а также учителям информатики, которые смогут затем передать эти знания своим ученикам) принципы обработки строковых данных;

• во-вторых, продемонстрировать применение некоторых возможностей Паскаля по работе со строками и символами, которых нет в Бейсике;

• в-третьих, показать преимущества использования такого редко используемого типа данных, как множества (как показывает практика, многие учителя попросту не знают, как можно использовать множества в реальном программировании, помимо “чисто теоретического” решения задач на пересечение, объединение и пр. множеств).

Итак, начнем...

В отличие от чисел (целых или вещественных, беззнаковых или со знаком) текст в компьютере, как все знают из курса информатики, представлен в виде последовательности кодов составляющих его символов — букв (латинских и строчных), знаков препинания, знаков математических операций и пр., а также специальных кодов, не имеющих отдельного визуально-

го представления в виде символов и служащих для управления размещением текста (например, это коды табуляции, перехода на новую строку и т.д.). При этом соответствие между конкретным символом и его кодом устанавливается согласно таблицам кодирования символов. Причем для символов национальных алфавитов (к которым относится и кириллица) могут использоваться различные 8-битовые таблицы кодирования (ASCII для MS-DOS, КОИ-8, Windows и др.), либо все такие символы объединены в 16-разрядной таблице кодирования стандарта Unicode.

Таким образом, каждый символ текста в памяти компьютера занимает один (или два — для Unicode) байт и хранится там в виде целого беззнакового числа. Поэтому, чтобы компьютер "не путал" их с обычными целыми числами, в языках программирования высокого уровня, как правило, для символьных и строковых типов данных предназначены отдельные, особые типы данных.

В языке Паскаль это:

• символьный тип **char**, предназначенный для представления одного какого-либо символа; символьная константа записывается в апострофах, например: '**a**', '**0**', '**+**' и т.д.;

• множественный (составной, структурированный, сложный) тип **string**, предназначенный для представления целых текстовых строк; строковая константа также записывается в апострофах, например: '**Строка**'.

При этом прослеживается иерархия типов: множественный тип **string** можно рассматривать как некий набор данных типа **char** (что отражает вполне очевидный факт — строка текста состоит из символов).

Отметим, что в языке Бейсик отдельный строковый тип данных тоже предусмотрен, но без разделения на символы и строки. Там просто считается, что отдельный символ — это строка, состоящая из одного символа. Как правило, для обозначения строкового типа данных используется записываемый после имени переменной, функции или процедуры символ "$".

Определение обоих этих типов данных (как и большинства других) в языке Паскаль производится в разделе **var**:

• для символьного типа данных —

var <*переменная*> **: char**;

• для строкового типа данных —

var <*переменная*> **: string**.

При этом строка может определяться как без указания ее размера (как продемонстрировано выше), так и с явным указанием ее длины (см. *рис.* 1):

var <*переменная*> **: string[**<*длина*>**];**

В подобном случае параметр <*длина*> представляет собой целое число, указывающее максимально допустимую длину строки, записываемой в такую переменную. Фактически же этот параметр указывает компьютеру, что для хранения такой переменной необходимо отвести указанное количество ячеек памяти для символов строки. Кроме них, в памяти также резервируется еще одна ячейка для хранения реальной длины строки, записанной в такую переменную: эта строка по длине может быть меньше, чем зарезервированная длина строковой переменной (и даже может быть пустой — не содержать символов вообще!), тогда часть зарезервированных ячеек памяти попросту не используется. А вот попытка записать в строковую переменную значение (строку), длина которой превышает объявленную длину строковой переменной, приведет к тому, что в этой переменной уместится только заявленное количество символов начала строки, а все остальное будет отброшено.

Следует также отметить, что в случае, когда мы не указываем размер определяемой в разделе **var** строковой переменной, его все равно определяет сама система программирования. В этом случае максимально допустимая длина строки составляет 255 символов, т.е. в памяти компьютера под такую строковую переменную резервируется 256 ячеек (одна, как и раньше, для хранения реальной длины хранящегося в этой переменной строкового значения). Все рассуждения о ситуациях, когда такой переменной присваивается более короткая или, наоборот, более длинная строка, тоже при этом остаются в силе.

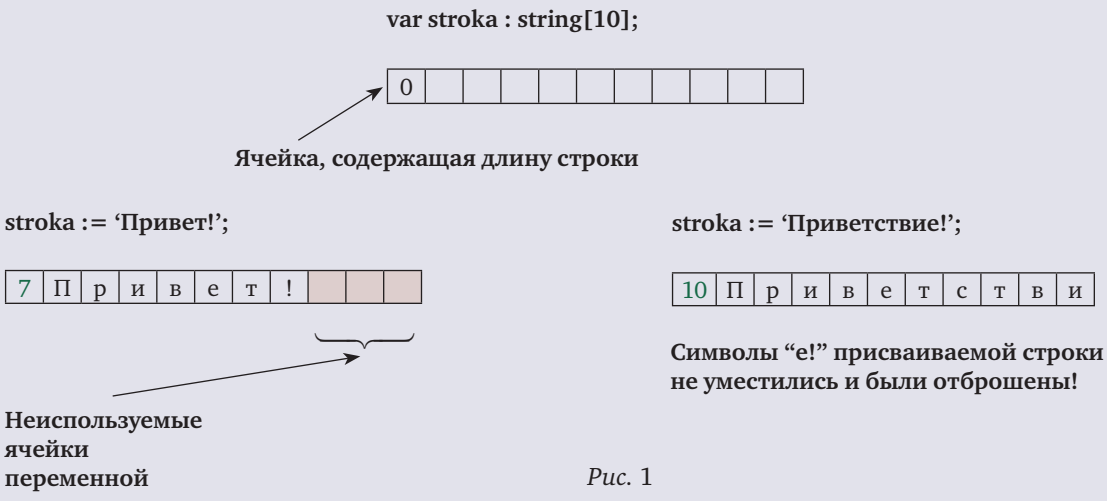


Рис. 1

Массивы символов и строк в языке Паскаль также возможны. Их определение и работа с ними осуществляются точно так же, как и с массивами чисел. Например, для описания одномерных массивов можно использовать команды:

• массив символов — **var** *<имя массива>* **: array[***<кол-во элементов>***] of char;**

• массив строк — **var** *<имя массива>* **: array[***<кол-во элементов>***] of string;**

При этом для строковых массивов можно после обозначения типа **string**, как и в случае простой строки, указать значение размера этих строк (все строки массива, разумеется, должны иметь одинаковый максимально допустимый размер).

Обращение же к элементам таких массивов производится полностью аналогично обращению к элементам числовых массивов — путем указания имени массива и записанного после него в квадратных скобках индекса элемента.

А теперь — внимание! — начинается самое интересное.

В отличие от Бейсика, где строки рассматриваются как "единое целое", в языке Паскаль (а также в ряде других языков, например, в Си) существует *дуализм представления строк*: к любой строке, определенной как тип **string**, можно в программе обращаться и как к единому целому (к переменной типа **string**), и как к одноименному одномерному массиву символов, указывая после имени строковой переменной в квадратных скобках номер (индекс) желаемого символа в строке! (При этом следует помнить, что символы в строках *всегда* нумеруются с единицы.)

Например, если мы определили строковую переменную оператором

var stroka : string[10];

и записали в нее строку '**Привет!**', то оператор

writeln(stroka[1]);

выведет на экран символ '**П**', а оператор

writeln(stroka[7]);

выведет символ восклицательного знака.

Соответственно, одномерный массив строк можно аналогичным образом рассматривать как двумерный массив символов, двумерный массив строк — как трехмерный массив символов и т.д.

Эта замечательная возможность делает ненужным использование функций извлечения символа из строки (типа **MID$(***<имя строки>***,***<позиция символа>***,1)** в Бейсике), поскольку для получения нужного символа достаточно просто обратиться к нему напрямую. Насколько это упрощает решение некоторых задач на работу со строками, мы увидим, когда перейдем к решению таких задач.

И наконец, завершая теоретический экскурс в символьные и строковые типы данных, перечислим некоторые имеющиеся в Паскале стандартные функции и процедуры для работы с символами и строками (информацию о них обычно можно найти в Help'е к используемой системе программирования, мы же будем рассматривать популярную в школьной практике среду программирования Pascal ABC.Net).

Chr(x)	Функция	Возвращает символ (типа **char**) по заданному 8-битному коду (типа **byte**) в кодировке Windows
ChrUnicode(x)	Функция	Возвращает символ (типа **char**) по заданному 16-битному коду (типа **word**) в кодировке Unicode
Ord(c)	Функция	Возвращает 8-битный код символа (типа **byte**) в кодировке Windows по заданному символу (типа **char**)
OrdUnicode(c)	Функция	Возвращает 16-битный код символа (типа **word**) в кодировке Unicode по заданному символу (типа **char**)
UpperCase(c) **UpCase(c)** **UpperCase(s)**	Функция	Возвращает символ (типа **char**) либо строку (типа **string**), преобразованные в прописной (заглавный, верхний) регистр (два варианта записи имени функции реализованы для обеспечения совместимости с другими различными версиями Паскаля)
LowerCase(c) **LowCase(c)** **LowerCase(s)**	Функция	Возвращает символ (типа **char**) либо строку (типа **string**), преобразованные в строчный (нижний) регистр
Pos(subs,s)	Функция	Возвращает номер позиции (число типа **integer**) в строке *s* (типа **string**), с которой в этой строке содержится подстрока *subs* (типа **string**). Фактически это — поиск вхождения подстроки в строку. Если такое вхождение не найдено, то возвращается нулевое значение (которое можно использовать как "флаг")
PosEx(subs,s,from)	Функция	Также возвращает номер позиции (число типа **integer**) в строке *s* (типа **string**), с которой в этой строке содержится подстрока *subs* (типа **string**), но поиск вхождения подстроки в строку ведется не с ее начала, а начиная с заданной позиции *from* (число типа **integer**). То есть возможные вхождения подстроки *до* указанной позиции будут проигнорированы. Если такое вхождение не найдено, то также возвращается нулевое значение (которое можно использовать как "флаг")
Length(s)	Функция	Возвращает целое число (типа **integer**), указывающее реальную длину текстовой строки (типа **string**), записанной, например, в некоторую строковую переменную *s*

SetLength(s, n);	Процедура	Устанавливает для заданной текстовой строки (типа **string**) новое значение длины (целое число типа **integer**)
Insert(source, s, index);	Процедура	Изменяет заданную строку *s* (типа **string**), вставляя в нее заданную подстроку *source* (также типа **string**), начиная с позиции *index* (целое число типа **integer**). Важно помнить, что реальная длина строки *s* при этом увеличивается
Delete(s,index,count);	Процедура	Изменяет заданную строку *s* (типа **string**), удаляя из нее часть символов (количество задается параметром *count* — целое число типа **integer**), начиная с позиции *index* (также целое число типа **integer**). Следует помнить, что реальная длина строки *s* при этом уменьшается
Copy(s,index,count)	Функция	Возвращает подстроку (типа **string**) длиной *count* (целое число типа **integer**), начинающуюся с позиции *index* (также целое число типа **integer**)
Concat(s1,s2,...)	Функция	Возвращает строку (типа **string**), "склеенную" (операция конкатенации) из двух или более заданных строк *s*1, *s*2, ... (типа **string**). То же самое можно сделать и при помощи операции конкатенации строк, записываемой знаком "**+**"
StringOfChar(c,count)	Функция	Возвращает строку (типа **string**), составленную из заданного количества (*count* — целое число типа **integer**) заданных символов (типа **char**). Например, функция **StringOfChar('*',5)** вернет строку "*****"
ReverseString(s)	Функция	Возвращает строку (типа **string**), в которой порядок следования символов изменен на обратный. Например, функция **ReverseString("привет")** вернет строку "**тевирп**"
CompareStr(s1,s2)	Функция	Сравнивает заданные строки (типа **string**) и возвращает числовое значение (типа **integer**): отрицательное, если $s1 < s2$, положительное, если $s1 > s2$, или равное нулю, если $s1 = s2$. Напомним, что строки можно сравнивать и "напрямую", записывая в условном операторе имена строковых переменных и требуемые знаки логических операций
LeftStr(s,count)	Функция	Возвращает подстроку (типа **string**) длиной *count* (целое число типа **integer**), начинающуюся с начала исходной строки (слева, с позиции 1)
RightStr(s,count)	Функция	Возвращает подстроку (типа **string**) длиной *count* (целое число типа **integer**) из конца исходной строки (справа)
Trim(s)	Функция	Возвращает строку (типа **string**) с удаленными из нее (из ее исходного вида) пробелами в начале и в конце (пробелы внутри строки не удаляются)
TrimLeft(s)	Функция	Возвращает строку (типа **string**) с удаленными из нее (из ее исходного вида) пробелами в начале (пробелы в конце и внутри строки не удаляются)
TrimRight(s)	Функция	Возвращает строку (типа **string**) с удаленными из нее (из ее исходного вида) пробелами в конце (пробелы в начале и внутри строки не удаляются)
StrToInt(s) **StrToInt64(s)** **StrToFloat(s)**	Функция	Данные функции преобразуют строку (типа **string**), содержащую символьную запись числа (цифры, точку, знак минуса) в собственно число типа **integer**, **int64** или **real** и возвращают это число. Распознавание числа выполняется настолько, насколько это возможно
TryStrToInt(s,value) **TryStrToInt64(s,value)** **TryStrToFloat(s,value)**	Функция	Данные функции также служат для преобразования строки (типа **string**), содержащей символьную запись числа (цифры, точку, знак минуса) в собственно число типа **integer**, **int64**, **single** или **real**, но работают несколько иначе. Если такое преобразование выполнено успешно, то число записывается в качестве значения параметра *value* (он должен иметь соответствующий тип), а функция возвращает логическое (типа **boolean**) значение **True**. Иначе (если в строке *s* содержится не запись числа) функция просто возвращает логическое значение **False**
Val(s,value,err);	Процедура	Служит для той же цели, что и описанные выше функции, — пытается преобразовать строку (типа **string**), содержащую символьную запись числа, в само число. Результат такого преобразования — число типа, который имеет параметр *value* (**byte**, **word**, **longword**, **integer**, **shortint**, **smallint**, **int64**, **uint64**, **single** или **real**), — при успехе записывается в качестве значения параметра *value*, а значение параметра *err* (типа **integer**) приравнивается нулю. Если же такое преобразование выполнить не удалось, то значение параметра *err* будет больше нуля
Str(x,s);	Процедура	Преобразует заданное число *x* (типа **integer** или **real**) в строку (типа **string**), содержащую символьную запись этого числа (цифры, точку, знак минуса)
IntToStr(x) **FloatToStr(x)**	Функция	Выполняют аналогичную операцию, возвращая для заданного числа *x* (типа **integer**, **int64** или **real**) в строку (типа **string**), содержащую символьную запись этого числа

При использовании этих функций и процедур необходимо помнить следующее:

• функции *возвращают* некоторое значение, которое нужно куда-то записать или как-то использовать, поэтому функцию надо записывать или в операторе присваивания (например: **d := Length(s);**), или, скажем, в операторе вывода на экран (**writeln(Length(s));**);

• процедуры в отличие от функций сами по себе не возвращают значений, а изменяют значение некоторых заданных в них параметров (аргументов), поэтому в составе оператора присваивания или вывода на экран их записывать нельзя. Процедура записывается отдельным оператором, например: **Delete(stroka,5,1);** — правильная запись (из строки *stroka* удаляется один символ, стоящий в 5-й позиции), а **stroka1 := Delete(stroka,5,1);** — неправильная запись.

А теперь перейдем, наконец, к решению задач.

Задача 1. Дана строка символов. Удалить из нее первый знак препинания.

Наиболее простое решение: определить длину введенной строки, реализовать цикл перебора всех ее символов (с первого до последнего, имеющего номер, равный значению длины), каждый очередной символ сравнивать с каждым из возможных символов — знаков препинания (“.”, “,”, “;”, “!” и т.д.) и при выполнении этого условия каким-то способом убрать его из строки, а затем — прервать цикл просмотра символов.

Реализуем эту идею на Паскале:

```
program z1;
var st, st1 : string;
    dl, i, k : integer;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  k := 0;
  for i := 1 to dl do
    if (st[i] = '.') or (st[i] = ',') or
       (st[i] = ';') or (st[i] = '!') or
       (st[i] = '?.') then begin
      k := i;
      break;
    end;
  if k <> 0 then st1 := LeftStr(st, k-1) +
                        RightStr(st, dl-k)
            else st1 := 'Знаков
                          препинания нет';
  writeln(st1);
end.
```

Проанализируем этот листинг.

1. Вводится строка и определяется ее длина *dl* (при помощи стандартной функции **Lenght**).

2. Переменная *k*, которая у нас одновременно будет служить для запоминания позиции найденного первого знака препинания и играть роль “флага”, обнуляется.

3. Строится цикл **for** перебора значения переменной *i* от 1 до значения длины строки *dl*.

4. В теле цикла мы должны извлечь очередной (записанный в позиции *i*) символ строки. И вот здесь проявляется удобство “дуализма” обращения к строкам в языке Паскаль: вместо того чтобы, как в Бейсике, записывать каждый раз функцию, извлекающую нужный символ как подстроку (в Бейсике — **MID$(ST$,I,1)**, в Паскале — **Copy(st,i,1)**), мы можем просто обратиться сразу к требуемому символу как к элементу массива *st* с индексом *i*: **st[i]**.

5. Очередной символ (**st[i]**) нужно сравнивать с каждым из возможных символов — знаков препинания, записывая операции сравнения типа **st[i] =** '.' через логическую связку **or** в операторе **if**. Тогда в ветви **then** (т.е. если очередной символ строки равен хотя бы одному знаку препинания) мы запоминаем его номер позиции (*i*) в переменной *k* и прерываем цикл оператором **break**.

6. После завершения цикла — досрочного по **break** или “штатного”, когда завершен перебор всех символов строки, а знак препинания в ней не найден, нам надо разделить эти два случая. Для этого мы используем “флаговую” функцию переменной *k*:

• если *k* не равно нулю, значит, знак препинания найден и его номер позиции в строке записан в *k*, — тогда мы выполняем операцию “удаления” этого *k*-го символа из строки (п. 7);

• иначе, если $k = 0$, это означает, что знак препинания найден не был, цикл завершился сам по себе, а значение *k* сохранилось исходное, которое мы присвоили этой переменной еще до цикла, — тогда мы просто должны вывести сообщение, что знаков препинания в строке нет.

7. Чтобы “удалить” найденный знак препинания, сделаем следующее. Оставляя исходную строку неизменной, будем формировать из нее новую строку. Сначала запишем в нее все символы исходной строки с первого до *k*-го (не включая его), а затем допишем (конкатенируем) к ней символы из правой части исходной строки от *k*-го (опять же не включая его) до последнего. Первую часть строки (слева от знака препинания) можно получить, используя функцию **LeftStr(st, k-1)**, а вторую (правую) — используя функцию **RightStr(st, dl-k)**. При этом вычисление количеств извлекаемых символов достаточно очевидно, если представить строку наглядно, как на *рис.* 2.

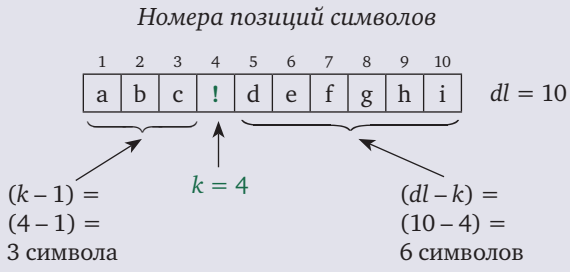


Рис. 2

Решение достаточно простое. Но запись условного оператора получается довольно громоздкой: ведь в нем надо перебрать (через **or**) все возможные случаи равенства очередного символа какому-нибудь знаку препинания. Кроме того, подумайте, что было бы, если бы таких сравнений требовалось несколько в разных местах программы и вдруг выяснилось бы, что надо изменить (скажем, дополнить знаком двоеточия) перечень обрабатываемых знаков препинания? Пришлось бы внимательно (но все равно с риском где-то что-то пропустить) просматривать всю программу, выискивать в ней все такие операторы сравнения и дописывать в них еще одно логическое условие…

Можно ли упростить программу, а заодно — и возможные модификации перечня знаков препинания, обрабатываемых в ней? Можно! И в этом нам поможет интересный, но, к сожалению, редко используемый множественный тип данных — *множество* (приносим извинения за получившийся каламбур ☺).

Множество — это набор однотипных элементов. Однако в отличие от массива, в котором такие элементы располагаются последовательно и пронумерованы индексами, множество — это просто группа элементов, “сваленных в одну кучу”. Индексов у элементов множества нет. Более того — для множества порядок записи в нем элементов не важен, например, [1, 2, 3] и [3, 2, 1] — это одно и то же множество цифр. Другая особенность множества — это уникальность его элементов: каждый из них должен присутствовать в множестве только “в одном экземпляре”, без повторов, — например, набор [1, 2, 3, 2, 1] множеством не является.

В языке Паскаль множества определяются следующим образом:

type <*имя типового множества*> **= set of** <*тип или набор элементов*>**;**

var <*имя экземпляра множества*> **:** <*имя типового множества*>**;**

То есть, сначала объявляется некий “класс” множеств, скажем, множество символов или чисел, а затем создается сколько угодно конкретных множеств этого типа. Например:

type mn1 = set of byte; — множество всех возможных беззнаковых целых 8-разрядных чисел;

type mn2 = set of char; — множество всех возможных символов;

type mn3 = set of ‘1’..‘9’; — множество всех цифр, кроме нуля;

type mn4 = set of ‘a’..‘z’; — множество всех латинских строчных букв.

А после определения типового множества при объявлении экземпляра можно задать требуемые значения элементов, например:

var mn : mn2 := [‘а’, ‘е’, ‘ё’, ‘и’, ‘о’, ‘у’, ‘ы’, ‘э’, ‘ю’, ‘я’]; — создаем множество всех русских строчных гласных букв.

Множество может оставаться и пустым (не содержать никаких элементов), тогда оно обозначается как **[]**.

Над множествами можно выполнять следующие операции:

• объединение (операция **+**) — результатом является множество, включающее (по одному разу!) элементы, которые есть хотя бы в одном из исходных множеств;

• пересечение (операция *) — результатом является множество, включающее только элементы, которые есть в каждом из исходных множеств;

• разность (операция **–**) — результатом для двух исходных множеств является множество, включающее только элементы, которые есть в первом из этих множеств, но отсутствуют во втором;

• проверка вхождения элемента в множество (операция **in**) — если данный элемент (левый операнд) входит в заданное множество (правый операнд), то возвращается результат **True** (логический тип **boolean**), иначе — результат **False**;

• добавление элемента в множество — операция типа **M := M + [**<*элемент*>**]** (где *M* — множество) либо процедура **include(**<*множество*>**,**<*элемент*>**)**;

• исключение элемента из множества — операция типа **M := M – [**<*элемент*>**]** (где *M* — множество) либо процедура **exclude(**<*множество*>**,**<*элемент*>**)**.

А теперь давайте посмотрим, как можно ту же самую задачу с удалением из строки первого знака препинания решить, используя множества.

Задача 1. Дана строка символов. Удалить из нее первый знак препинания.

```
program z1;
type znak = set of char;
var st, st1 : string;
    dl, i, k : integer;
    zn : znak = ['.', ',', ';', '!', '?'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  k := 0;
  for i := 1 to dl do
    if st[i] in zn then begin
      k := i;
      break;
    end;
  if k <> 0 then
     st1 := LeftStr(st, k-1) +
            RightStr(st, dl-k)
  else st1 := 'Знаков препинания нет';
  writeln(st1);
end.
```

Строки листинга, которых не было в предыдущем варианте решения или которые были изменены по сравнению с ним, выделены жирным шрифтом и синим цветом.

Вначале мы определяем типовое множество *znak* как множество всех возможных символов.

Затем, в разделе **var**, мы объявляем экземпляр такого множества — *zn* — и заносим в него элементы, которые представляют собой символы всех возможных знаков препинания.

Наконец, в операторе **if** (в цикле перебора символов заданной текстовой строки) нам достаточно записать проверку вхождения очередного символа в заданное нами множество знаков препинания: **if st[i] in zn then ...** А далее обработка найденного знака препинания производится так же, как и в предыдущей программе. Впрочем, можно взамен записать и соответствующую процедуру — это будет короче:

```
...
  if k <> 0 then Delete(st,k,1)
            else st := 'Знаков
                       препинания нет';
```

Только в этом случае придется отказаться от формирования новой строки *st*1 и вместо этого изменять исходную строку *st* (соответственно, надо записать именно ее и в "итоговом" операторе вывода полученной строки на экран).

Ну как? Очевидно, что запись условного оператора, проверяющего, является ли очередной символ знаком препинания, стала гораздо компактнее и проще!

Но это — только один из получаемых нами благодаря использованию множества "плюсов". Если теперь нам надо дополнить список обрабатываемых знаков препинания двоеточием, то достаточно во всей программе (сколько бы в ней ни было проверок на соответствие символов одному из знаков препинания) дополнить только одну строку:

```
zn : znak = ['.', ',', ';', '!', '?', ':'];
```

Красиво и просто получается, не правда ли?

А мы тем временем рассмотрим другие задачи.

Задача 2. Дана строка символов. Удалить из нее *все* знаки препинания.

Очевидно, что от предыдущей задачи она отличается тем, что нам после обнаружения и удаления первого знака препинания надо не прерывать цикл, а продолжить просмотр строки до конца. Но если мы просто уберем оператор **break** и перенесем операцию удаления найденного символа из строки внутрь цикла **for**, то потерпим полное фиаско — вместо правильного ответа получим сообщение об ошибке в процессе выполнения из-за выхода за пределы массива. Почему?

Причина станет понятной, если задуматься над тем, как меняется наша строка при удалении из нее очередного знака препинания: эта строка *становится короче* на один символ! А у нас интервал изменения цикловой переменной в **for** задается "раз и навсегда" — от 1 до *прежнего*, исходного значения длины строки *dl*. Например, было в ней сначала 10 символов — тогда цикл будет задан по *i* от 1 до 10; нашли и удалили в строке знак препинания — реальная длина строки уменьшилась до 9, а цикл-то по прежнему будет выполняться, пока *i* не станет равно 10! И как только мы попробуем выполнить обращение к символу **st[10]**, произойдет ошибка: такого символа не существует.

Как быть? Очевидно, вместо **for** придется реализовать цикл **repeat**, а еще лучше — **while**, который сразу проверяет условие выполнения цикла, поскольку в них мы сможем гибко отслеживать изменение длины строки и реальное достижение ее последнего символа.

```
program z_4_4_2;
type znak = set of char;
var st : string;
    dl, i : integer;
    zn : znak = ['.', ',', ';', '!', '?'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  i := 1;
  while i <= dl do begin
    if st[i] in zn then begin
      Delete(st,k,1);
      dl := dl - 1;
    end;
    i := i + 1;
  end;
  writeln(st);
end.
```

Сделаем несколько комментариев.

1. Здесь мы опять-таки используем множество символов — знаков препинания.

2. Поскольку нам не нужно проверять, закончился ли цикл по **break** или "штатно", нам не нужна переменная *k*, — мы можем работать непосредственно с цикловой переменной *i*.

3. Если знак препинания найден, то сразу после его удаления (процедурой **Delete**) необходимо соответственно уменьшить значение длины строки *dl* на единицу.

4. Поскольку в отличие от **for** в цикле **while** изменение значения цикловой переменной не выполняется автоматически, надо в конце тела цикла прибавить 1 к *i*, а также еще перед циклом задать исходное значение $i = 1$ путем присваивания.

Как работает эта программа?

Предположим, что исходная строка имела длину 6 символов и содержала два знака препинания. Первоначально цикловое условие тогда будет таким: $i \le 6$, — оно истинно, и очередной проход цикла выполняется. Если бы в строке не было ни одного знака препинания (и, соответственно, ее длина оставалась бы постоянной), то цикл так и "пробежал" бы все значения *i* до 6 включительно, после чего условие цикла стало бы ложным (*i* стало бы равно 7 в начале очередного прохода цикла) и цикл бы благополучно завершился. Но вот мы нашли первый из двух знаков препинания и

удалили его. Длина строки *st* уменьшилась на 1. Но! Мы уменьшили на 1 и значение *dl*, с которым в цикловом условии сравнивается *i*. А после того как мы найдем и удалим из строки второй знак препинания, и реальная длина строки *st*, и значение *dl* станут оба равны 4. В результате наше цикловое условие будет оставаться корректным, несмотря на изменение длины строки *st* при удалении любого числа символов из нее.

Сделаем важный практический вывод. Если в процессе обработки строк их длина меняться не должна (например, надо всего лишь заменить один символ на другой) либо если требуется найти и удалить только один символ и прервать после этого просмотр остальной части строки, то можно (и это удобнее) использовать цикл **for**. Если же длина строки при ее обработке меняется (из нее удаляются или в нее вставляются дополнительные символы), то придется использовать цикл **while**.

Задача 3. Дана строка символов. Заменить в ней все знаки "!" на точки.

После всего сказанного выше думается, что читатели могут решить эту задачу самостоятельно (решение дано в конце статьи).

Справились? Тогда усложним эту задачу.

Задача 3*а*. Дана строка символов. Заменить в ней все знаки препинания на пробелы.

Впрочем, по аналогии с ранее решенными задачами читатели наверняка найдут решение и здесь.

А теперь — задача, наверное, самая сложная из всех.

Задача 4. Дана строка символов. Повторить в ней все знаки препинания трижды.

Решить ее мы также предлагаем самим читателям. А мы продолжим решение задач на символы и строки, сосредоточив свое внимание на строковых функциях.

Задача 5. Дана строка символов. Выделить из нее подстроку между первой и второй точками.

Такая задача на первый взгляд кажется сложной, но благодаря имеющемуся в Паскале достаточно богатому набору строковых функций она решается буквально "в два действия". Учитывая, что читатели уже несколько "поднаторели" в работе со строками и символами, сразу приведем листинг программы — решения задачи:

```
program z_4_4_7;
var st : string;
    t1, t2 : integer;
begin
  writeln('Введите строку');
  readln (st);
  t1 := Pos('.',st);
  t2 := PosEx('.',st,t1+1);
  if (t1 = 0) or (t2 = 0) then
     st := 'В строке нет двух точек!'
  else st := Copy(st,t1+1,t2-t1-1);
  writeln(st);
end.
```

Думается, принцип ее решения прост и понятен.

Сначала мы при помощи функции *Pos* (поиск вхождения подстроки в строку) ищем номер позиции первой точки (напомним, что если точка в строке не найдена, то будет возвращено значение *t*1, равное нулю).

Затем нам надо найти номер позиции второй по счету точки. Чтобы при этом уже найденная первая точка нам не мешала, мы используем уже функцию *PosEx* для поиска вхождения точки в строку, начиная с позиции, следующей после уже найденной первой точки (т.е. с позиции *t*1 + 1).

А теперь остается только проверить: если хотя бы одно из значений — *t*1 или *t*2 — равно нулю, то, соответственно, в строке не найдена первая или вторая точка, и мы в качестве ответа заносим в строку *st* (вместо ее прежнего содержимого) сообщение об этом. Иначе (если оба значения позиций ненулевые — две точки найдены) мы используем функцию извлечения подстроки из строки *Copy*. В ней (напомним) надо указать позицию, с которой начинается извлекаемая строка, и количество извлекаемых символов. Начальная позиция определяется очевидно: это позиция символа, следующего после первой из найденных точек, т.е. *t*1 + 1. А как вычислить количество извлекаемых символов? Посмотрим на *рис.* 3: поскольку нам нужны только символы между найденными позициями (точками), но не сами эти точки, искомое количество символов на единицу меньше, чем разность значений *t*2 и *t*1.

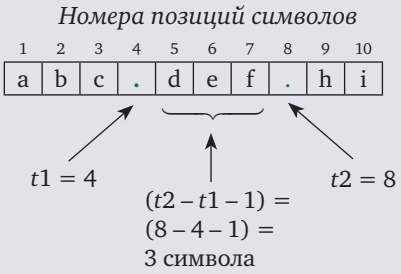


Рис. 3

А теперь попробуйте сами решить эту же задачу, но без использования строковых функций *Pos*, *PosEx* и *Copy*.

Решили? Тогда рассмотрим задачу более сложную.

Задача 6. Дана строка символов. Выделить из нее подстроку между первой и последней точками.

Все, что здесь изменилось, — то, что нужно вместо второй найти позицию последней точки. Но как это сделать? Если бы в Паскале имелась функция, аналогичная *Pos*, но просматривающая строку не слева направо, а справа налево, то данная задача решалась бы так же просто, как и предыдущая. Но такой "правосторонней" функции, увы, нет. Функция же *PosEx* тоже нам не подходит: ведь мы не знаем, с какой позиции строки надо начать просмотр. Как быть?

Можно искать нужную нам последнюю точку “вручную”: запустить цикл перебора символов от позиции *dl* до позиции *t*1 + 1 (так как уже найденная первая точка нам не нужна), проверять каждый символ на совпадение с ‘.’ и при успехе запомнить найденную позицию в *t*2 и прервать цикл. Но мы взамен применим небольшую хитрость.

Зеркально отразим исходную строку, воспользовавшись функцией `ReverseString()`. Тогда нужная нам последняя точка станет первой, и мы легко найдем ее позицию при помощи той же самой функции *Pos* и запишем в переменную *t*2. А теперь снова зеркально отразим строку с помощью функции `ReverseString()`. Поскольку эта операция обратима, мы снова получим исходную строку. А что теперь означает содержимое *t*2? Очевидно, это номер позиции последней точки при ее отсчете справа налево! А что надо сделать, чтобы получить (зная общую длину строки) номер этой позиции считая слева? Правильно: вычесть значение *t*2 из значения длины строки и прибавить единицу (*рис.* 4).

Рис. 4

Остается только учесть еще одно обстоятельство. При проверке — найдены ли в строке две требуемые точки — нам надо отследить два “ошибочных” случая: если в строке нет ни одной точки вообще или если в ней есть всего одна точка. Первый случай отследить легко — тогда значение *t*1 будет нулевым. А если точка в строке всего одна, то именно она будет найдена и в первом, и во втором поиске, и тогда *t*1 будет равно *t*2. Это и есть наше второе условие проверки.

Тогда наша программа будет иметь следующий вид:

```
program z_4_4_8;
var st : string;
    dl, t1, t2 : integer;
begin
  writeln('Введите строку');
  readln (st);
  t1 := Pos('.',st);
  st := ReverseString(st);
  t2 := Pos('.',st);
  st := ReverseString(st);
  dl := Length(st);
  t2 := dl-t2+1;
  if (t1 = 0) or (t2 = t1)
  then st := 'В строке нет двух точек!'
  else st := Copy(st,t1+1,t2-t1-1);
  writeln(st);
end.
```

Еще две задачи на работу со строковыми данными могут оказаться как очень простыми, так и довольно сложными в решении, — все зависит от того, какой “программный инструментарий” мы будем при этом использовать.

Задача 7. Дана строка символов. Вывести ее на экран задом наперед.

Вспомним, что среди изученных нами стандартных процедур и функций языка PascalABC.Net, предназначенных для обработки строковых данных, есть такая интересная функция, как **ReverseString()**, которая как раз и выполняет нужное нам действие — “выворачивает наизнанку” (вернее — задом наперед) заданную текстовую строку. Тогда рассматриваемая задача решается ***очень*** просто:

```
program z_4_4_15a;
var st : string;
begin
  writeln('Введите строку');
  readln (st);
  st := ReverseString(st);
  writeln(st);
end.
```

Но вот в других языках программирования (и даже в других версиях Паскаля) такой полезной функции может и не быть. Как тогда?

Задача 8. Дана строка символов. Вывести ее на экран задом наперед. Функцию реверса не использовать.

Если вспомнить, что в Паскале строковые данные “двулики”, как мифический древнеримский Янус, и могут рассматриваться и как текстовая строка, и как одномерный массив символов, из которых эта строка состоит, то можно предложить следующее простое решение такой задачи. При этом мы, однако, идем на небольшую “хитрость”, дословно понимая формулировку условия задачи: согласно ей, нам достаточно просто вывести строку на экран “наоборот”. А это нетрудно сделать, просто перебирая символы строки сзади наперед при помощи соответствующего цикла и печатая на экране в одну строку эти символы:

```
program z_4_4_15b;
var st : string;
    dl, i : integer;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  for i := dl downto 1 do write(st[i]);
  writeln;   // перевести строку
end.
```

Однако же для таких хитрых, как мы, можно сделать условие задачи более строгим.

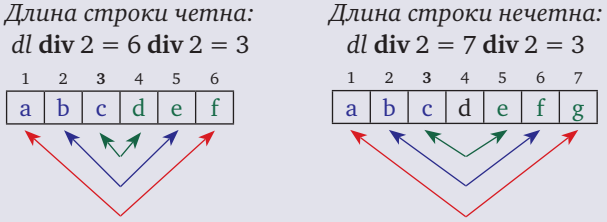
Задача 9. Дана строка символов. Преобразовать ее задом наперед до вывода на экран. Функцию реверса не использовать.

Теперь наша “хитрость” уже не проходит: нам нужно “перевернуть” строку задом наперед именно в памяти ПК. Как это сделать?

Проще всего — попарно менять местами символы строки (которая здесь опять-таки рассматривается как одномерный массив символов). Однако при этом нужно быть внимательными при определении границ изменения цикловой переменной. Построить цикл от первого до последнего символа нельзя.

(При объяснении этого материала учащимся можно остановиться и предложить им самим подумать — почему? Правильный ответ: ведь тогда мы сначала поменяем местами, например, первый символ с последним (в начале цикла при $i = 1$), а потом, перед окончанием цикла, когда $i = dl$, мы последний символ поменяем местами с первым. А в итоге символы в каждой паре поменяются местами дважды, и строка, которая было приняла требуемый нам “вывернутый” вид, снова превратится в исходную.)

Поэтому нужно выполнять цикл для значений i, “пробегающих” только половину строки: если ее длина — четное число, то перебираем символы с начала строки до ее середины, а если длина строки нечетна, то от начала до среднего символа, не затрагивая его (*рис.* 5). Очевидно, что оба этих случая можно реализовать одним и тем же циклом, где i изменяется от 1 до dl **div** 2 (целочисленное деление).



Меняем местами:
- элемент 1 ⇔ элемент *dl*
- элемент 2 ⇔ элемент *dl* – 1
- элемент 3 ⇔ элемент *dl* – 2

Рис. 5

```
program z_4_4_15c;
var st : string;
    dl, i : integer;
    buf : char;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  for i := 1 to dl div 2 do begin
    buf := st[i];
    st[i] := st[dl-i+1];
    st[dl-i+1] := buf;
  end;
  writeln(st);
end.
```

(Для обмена символов местами мы используем “буферную” переменную символьного типа.)

А вот еще одна задачка. Напомним, что *палиндром* — это такая строка, которая одинаково читается как справа налево, так и слева направо. Однако в нашем, “программистском” понимании в отличие от общепринятого строка-палиндром обладает двумя важными отличиями:

1) это может быть не обязательно какой-то осмысленный текст, а вообще любой набор символов, в том числе, например, запись числа;

2) если обычно в палиндроме пробелы не учитываются, то в нашем случае они важны, равно как и регистр символов (заглавный/строчный). Скажем, всем известная строчка “А роза упала на лапу Азора” в нашем понимании не является палиндромом (либо ее нужно записать в едином регистре и без пробелов: “арозаупалаланалапуазора”). А, например, фраза (придуманная авторами статьи) “город рим — мир дорог” — это, безусловно, палиндром.

Задача 10. Дана строка символов. Проверить, является ли она палиндромом.

Для начала попробуем решить ее, считая допустимым использование любых стандартных процедур и функций.

В этом случае решение следует из самого определения палиндрома — “одинаково читается как слева направо, так и справа налево”: достаточно сравнить саму строку с ее “вывернутой задом наперед” версией, получаемой при помощи функции **ReverseString()**:

```
program z_4_4_16a;
var st : string;
    dl : integer;
begin
  writeln('Введите строку');
  readln (st);
  if st = ReverseString(st) then
     writeln(st, ' - палиндром')
  else writeln(st, ' - не палиндром');
end.
```

Не правда ли — коротко и изящно? ☺

А теперь решим ту же самую задачу, не используя “волшебную” функцию реверса.

Задача 11. Дана строка символов. Проверить, является ли она палиндромом. Функцию реверса не использовать.

Помня решение предыдущей задачи (где требовалось получить “реверсированную” строку), несложно догадаться, что и здесь нам потребуется попарно перебирать символы строки (рассматриваемой как одномерный массив символов) от ее начала и конца к середине. Да и цикл будет строиться точно так же — от 1 до dl **div** 2. Однако теперь нам надо не переставлять символы, а сравнивать их между собой. При этом, чтобы сделать вывод, что заданная строка — не палиндром, достаточно обнаружить хотя бы одно несовпадение символов в какой-нибудь их паре. Поэтому в качестве “фла-

га” удобно использовать логическую (тип **boolean**) переменную:

• вначале ей присваивается значение **true** (предполагаем, что строка является палиндромом);

• при обнаружении на каком-то шаге цикла несовпадения символов переменная-“флаг” приравнивается значению **false** (и далее уже никакие совпадения или несовпадения ничего не изменят, можно даже прервать цикл командой **break**);

• по окончании цикла проверяем состояние “флага” и, если он по-прежнему равен **true**, выводим сообщение, что строка является палиндромом, а если **false** — что это не палиндром.

```
program z_4_4_16b;
var st : string;
    dl, i : integer;
    fl : boolean;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  fl := true;
  for i := 1 to dl div 2 do
    if st[i] <> st[dl-i+1] then
       fl := false;
  if fl then writeln(st, ' - палиндром')
     else writeln(st, ' - не палиндром');
end.
```

И наконец, “на закуску”, рассмотрим еще несколько задач, в которых от нас потребуется умение работать и с символьными, и с числовыми данными, а также умение преобразовывать их друг в друга.

Задача 12. Дана строка символов. Определить, является ли она записью четного десятичного числа.

Речь здесь идет о том, что заданная строка текста состоит из символов цифр, т.е. является текстовой записью некоторого числа. Однако при этом числом она с точки зрения компьютера не является! Вспомним, что в памяти ПК число (например, типа **integer** — целое со знаком, одинарной точности) хранится в двоичном формате и занимает одну двухбайтную ячейку ОЗУ. А текстовая запись числа, о которой идет речь в этой задаче, хранится в памяти как последовательность кодов символов, соответствующих отдельным цифрам, где каждый код символа занимает один байт.

Среди стандартных функций и процедур языка Паскаль есть и такие, которые могут попытаться “распознать” в заданной строке текста запись числа. Такие функции или процедуры просматривают строку и, если в ней нет нецифровых символов (например, букв), то цифра за цифрой преобразуют запись числа в само число.

(Можно предложить учащимся подумать над тем, как такая функция работает. Самый простой алгоритм может быть таким. Вначале для хранения будущего числа объявляется и обнуляется числовая переменная. Строка просматривается слева направо символ за символом. Очередной ASCII-код символа проверяется на вхождение в диапазон от 48 до 57 (что соответствует символам цифр от “0” до “9”); при несоответствии работа программы прерывается с выдачей специального кода ошибки. Если же очередной символ — это цифра, то из его кода ASCII вычитается константа 48 (тогда мы получаем значение очередного разряда числа), текущее значение числовой переменной умножается на 10 и к нему прибавляется только что найденное значение очередного разряда.)

Решение же нашей задачи при использовании, например, процедуры **Val** для преобразования текстовой записи числа в целое число типа **integer** (поскольку в процедуре записан второй по счету аргумент именно этого типа) будет несложным: достаточно сначала преобразовать строку в число, а затем проверить это число на четность всем известным приемом — проверкой на равенство нулю остатка от деления числа на 2.

```
program z_4_4_13a;
var st : string;
    chis, err : integer;
begin
  writeln('Введите строку');
  readln (st);
  Val(st, chis, err);
  if err = 0 then begin
    if chis mod 2 = 0 then
      writeln(st,' - запись четного числа')
    else
     writeln(st,' - запись нечетного числа');
    end
   else writeln(st,' - не является целым числом!');
end.
```

(Третий параметр — *err* — это признак ошибки при преобразовании строки в число. Если такое преобразование выполнено успешно, то *err* = 0. Именно это мы проверяем, прежде чем начать проверку полученного числа на четность, а если это не так, то мы просто выводим сообщение, что заданная строка — не число.)

Однако, проверяя работу этой программы на различных входных данных, мы увидим, что она способна работать только со строками, не содержащими ни одного нецифрового символа и имеющими небольшую длину. И если первое понятно (процедура **Val** требует, чтобы в строке были только символы-цифры), то, например, причина вывода сообщения “не является целым числом” для входной строки 2222222222222222 может оказаться неясной.

(Предложите учащимся самим подумать над причиной, по которой наша программа “не признает” числа, записанные в длинных строках. Объяснение этому факту нужно искать в ограниченности разрядной сетки, отводимой под числа типа **integer**.)

А можно ли сделать программу более универсальной — такой, чтобы она могла распознавать число “внутри” заданной строки текста, даже когда перед и после записи числа имеются произвольные нецифровые символы? И чтобы при этом не было

ограничений на длину текстовой записи числа? Очевидно, в этом случае нам придется отказаться от использования стандартных функций и процедур преобразования строкового типа данных в числовой и все делать вручную.

Задача 13. Дана строка символов. Определить, является ли содержащаяся в ней запись четным десятичным числом.

Во-первых, нам потребуется просматривать строку и проверять каждый ее символ — является ли он записью цифры. Поэтому удобно будет определить в программе множество *cf* всех возможных цифр и проверять символы на их вхождение в это множество.

Во-вторых, поскольку запись числа может оказаться где-то внутри строки, нам потребуется фиксировать ее местоположение (для чего — будет сказано чуть ниже), поэтому заготовим в программе переменную-флаг *fl* логического типа.

Наконец, в-третьих, нужно подумать, как определять факт четности числа, — ведь теперь мы не будем преобразовывать текстовую запись содержащегося в строке числа в собственно число, а потому уже не можем использовать операцию вычисления остатка от деления (**mod**), так как к строкам она неприменима. Поэтому вспомним хорошо известный нам из математики признак делимости: число делится нацело на 2, если его последняя цифра делится на 2. Поэтому "ключевым" моментом нашего алгоритма будет: *найти в строке последнюю цифру содержащейся в ней записи числа и проверить, является ли эта цифра четной*. Проще всего сделать это опять-таки при помощи множеств — объявить экземпляр множества символов *cf*2 (по тому же самому "типовому" множеству *cifra*, которое уже создано нами в разделе **type**), "наполненный" только символами четных цифр, и проверять извлеченную из строки последнюю цифру числовой записи на вхождение в это "четное" множество *cf*2.

Листинг программы, реализующей этот алгоритм, может быть таким:

```
program z_4_4_13b;
type cifra = set of char;
var st : string;
    dl, i, k : integer;
    fl : boolean;
    cf : cifra = ['0','1','2','3','4',
                  '5','6','7','8','9'];
    cf2 : cifra = ['0','2','4','6','8'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  fl := false;
  for i := 1 to dl do begin
    if (fl = false) and (st[i] in cf) then
        fl := true;
    if (fl = true) and not(st[i] in cf) then
      begin
        fl := false; k := i-1; break;
      end;
  end;
  if fl = true then k := dl;
  if st[k] in cf2 then
     writeln(st,' содержит четное число')
  else writeln(st,' не содержит
               четного числа');
end.
```

Посмотрим, как работает эта программа.

Сначала мы определяем длину введенной строки (чтобы построить цикл перебора всех ее символов с первого до последнего) и присваиваем переменной-флагу *fl* изначальное значение *false* (предполагаем, что строка начинается не с символа цифры).

Далее в цикле мы на каждом шаге:

• проверяем: если флаг все еще равен *false* (т.е. до этого цифр в строке не было встречено), а очередной символ — это цифра, то меняем состояние флага на *true* (зафиксировано начало записи в этой строке числа);

• проверяем: если флаг уже равен *true* (т.е. мы перебираем символы цифр записи числа) и вдруг очередной просматриваемый символ в строке **не является** цифрой (**not(st[i] in cf)**), то это значит, что запись числа в строке закончилась на предыдущем, $(i-1)$-м символе.

Указанные два условных оператора позволяют нам, таким образом, найти в строке первую содержащуюся в ней запись числа и определить позицию в строке последнего символа цифры в этой записи (*рис.* 6). Тогда мы снова выставляем флаг *fl* в *false*, запоминаем в переменной *k* значение позиции предыдущего символа $(i-1)$ и прерываем цикл.

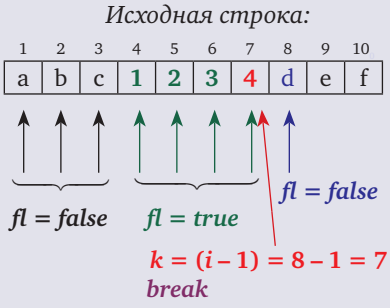


Рис. 6

• После цикла проверяем отдельным условным оператором: если *fl* все еще равен *true*, то это означает, что цикл закончился сам собой, без обнаружения первого нецифрового символа после записи числа и без досрочного прерывания цикла. А что это означает? Это значит, что наша строка записью числа и заканчивается, а последняя цифра этого числа находится в последнем символе нашей строки ($k = dl$).

• Ну а теперь, когда мы точно знаем, где в строке находится символ последней цифры записи числа, мы обращаемся к нему как к элементу символьного

массива и проверяем его на вхождение в множество четных цифр *cf2*: если он в это множество входит, то и записанное в строке число — четное, а иначе — оно нечетное.

Задача 14. Дана строка символов. Определить, является ли содержащаяся в ней запись десятичным числом, кратным 3.

Эта задача похожа на предыдущую: в ней нам тоже потребуется искать в заданной строке запись числа и анализировать ее символы-цифры. Но здесь нам нужно проверять кратность числа трем, что несколько сложнее: чтобы число делилось на 3 нацело, нужно, чтобы его сумма цифр делилась на 3. Поэтому если в предыдущей задаче нам достаточно было выделить и проверить только одну последнюю цифру, то теперь потребуется фиксировать начало и конец записи числа в строке, а затем подсчитать сумму всех этих цифр, превращая каждую из них отдельно в число.

```
program z_4_4_14;
type cifra = set of char;
var st : string;
    dl, i, k1, k2, summa : integer;
    fl : boolean;
    cf : cifra = ['0','1','2','3','4','5',
                  '6','7','8','9'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  fl := false;
  for i := 1 to dl do begin
    if (fl = false) and (st[i] in cf) then
    begin
      fl := true; k1 := i;
    end;
      if (fl = true) and not(st[i] in cf) then
    begin
      fl := false; k2 := i-1; break;
    end;
  end;
  if fl = true then k2 := dl;
  summa := 0;
  for i := k1 to k2 do
    summa := summa + StrToInt(st[i]);
  if summa mod 3 = 0 then
     writeln(st,' содержит число, кратное 3')
  else writeln(st,' не содержит числа,
               кратного 3');
end.
```

Разберем этот листинг.

Вначале, как и ранее, мы определяем длину введенной строки (чтобы организовать цикл перебора ее символов) и задаем исходное значение флага *fl* равным *false*.

Затем на каждом шаге первого цикла (“поискового”) мы:

• проверяем: если *fl* = *false* и при этом встретился символ-цифра, то это значит, что мы нашли начало записи числа, — “переключаем” флаг в *true* и запоминаем позицию текущего символа (первой цифры записи числа) в переменной *k*1;

• проверяем: если *fl* = *true* и при этом встретился символ — не цифра, то мы нашли конец записи числа, — “переключаем” флаг в *false*, запоминаем позицию *предыдущего* символа (последней цифры записи числа) в переменной *k*2 и прерываем цикл по **break** (*рис.* 7).

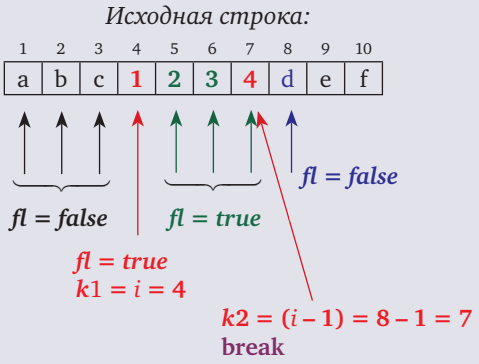


Рис. 7

После цикла отслеживаем случай, когда записью числа введенная строка и оканчивается: если по завершении цикла *fl* по-прежнему равно *true*, то последняя цифра записи числа — это последний символ строки (*k*2 = *dl*).

Теперь мы знаем, где в строке находится запись числа (от позиции *k*1 до позиции *k*2). Для подсчета суммы цифр этого числа обнуляем переменную *summa*, в которой будем накапливать эту сумму, и запускаем второй цикл (“расчетный”) по *i* от *k*1 до *k*2. На каждом его шаге мы извлекаем очередной символ — цифру из исходной строки, преобразуем его в число (функция **StrToInt()**) и прибавляем это число к переменной-“накопителю”. (Очевидно, что, поскольку мы работаем каждый раз только с одной цифрой, эту операцию можно выполнять без ограничений для достаточно длинных записей чисел.)

А дальше все просто: проверяем полученную сумму цифр на кратность 3 с помощью операции **mod** и выводим соответствующее сообщение.

А теперь — две задачи для самостоятельного решения.

Задача 15. Дана строка символов. Определить, является ли содержащаяся в ней запись десятичным числом, кратным 4.

Задача 16. Дана строка символов. Определить, является ли содержащаяся в ней запись десятичным числом, кратным 5.

Для подсказки напомним признак делимости числа на 4: “На 4 делятся все натуральные числа, две последние цифры которых составляют или нули, или число, кратное 4”. Что же касается решения задачи 16, то здесь читатели должны обо всем догадаться сами… ☺

Решение задачи 3

Задача очень простая. Длина строки в процессе ее обработки не меняется — значит, можно использовать цикл **for**. В нем достаточно проверять — равен ли очередной символ знаку '!', и если да, то выполнять переприсваивание этого очередного (*i*-го) символа, заменяя его на '.' .

> Кстати, — а вот если бы у нас не было возможности обращаться к символам строки напрямую как к элементам символьного массива, то реализовать эту операцию было бы заметно сложнее. Например, в Бейсике потребовалось бы написать:
>
> **IF MID$(ST$,I,1) = "!" THEN**
>
> **ST$ = LEFT$(ST$,I-1) + "." + RIGHT$(ST$,DL-I)**

```
program z3;
var st : string;
    dl, i : integer;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  for i := 1 to dl do
    if (st[i] = '!') then st[i] := '.';
  writeln(st);
end.
```

Решение задачи 3*a*

Эта задача решается точно так же, как задача 3, но теперь в ней потребуется использовать множество всех знаков препинания, а в операторе **if** не просто сравнивать очередной символ с '!', а проверять его на вхождение в заданное множество знаков препинания, и если дело обстоит именно так, то переприсваивать этот символ на пробел.

```
program z3a;
type znak = set of char;
var st : string;
    dl, i : integer;
    zn : znak = ['.', ',', ';', '!', '?', ':'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  for i := 1 to dl do
    if st[i] in zn then st[i] := ' ';
  writeln(st);
end.
```

Решение задачи 4

```
program z4;
type znak = set of char;
var st : string;
    dl, i : integer;
    zn : znak = ['.', ',', ';', '!', '?', ':'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  i := 1;
  while i <= dl do begin
    if st[i] in zn then begin
      st := LeftStr(st, i-1) +
            StringOfChar(st[i],3) +
            RightStr(st, dl-i);
      dl := dl + 2;
    end;
    i := i + 3;
  end;
  writeln(st);
end.
```

Эта задача во многом похожа на ту, где требовалось удалить из строки все знаки препинания.

Поскольку здесь вместо каждого знака препинания должно вставляться три, длина строки в процессе ее обработки изменяется (увеличивается), поэтому надо использовать цикл **while**, а не **for**.

В цикле мы, как и раньше, проверяем очередной символ **st[i]** на вхождение в заданное множество знаков препинания.

Если знак препинания обнаружен, то мы заменяем его на три таких знака следующим образом:

- берем левую часть исходной строки до найденного знака;
- дописываем к ней строку, полученную троекратным повторением найденного знака препинания (помните про функцию **StringOfChar(s,count)** ?);
- дописываем к полученному правую часть исходной строки после найденного знака препинания;
- все, что получилось в результате такой конкатенации, присваиваем исходной строке, заменяя ее;
- поскольку длина строки увеличилась на 2 (был один символ — стало три), нужно увеличить на 2 значение *dl*, используемое в цикловом условии как граница изменения *i*.

Наконец, на следующем шаге цикла мы должны рассматривать символ, который до замены найденного знака препинания был следующим после него. Но, вставив еще два знака перед ним, мы "оттеснили" этот следующий проверяемый символ на две позиции правее. Значит, надо увеличить значение *i* не на 1, как раньше, а на 3 (см. *рис.* 8).

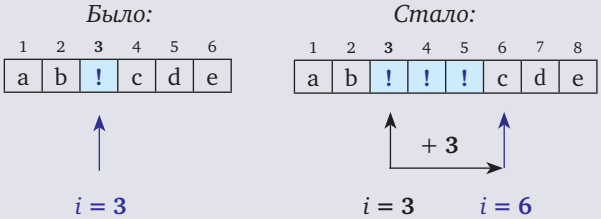


Рис. 8

Решение задачи 5

```
program z_4_4_7;
var st : string;
    dl, i, t1, t2 : integer;
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  t1 := 0; t2 := 0;
  for i := 1 to dl do
    if (st[i] = '.') and
       (t1 = 0) then t1 := i
        else if (st[i] = '.') then
          begin
            t2 := i; break;
          end;
  if (t1 = 0) or (t2 = 0) then
     st := 'В строке нет двух точек!'
  else begin
    for i := t1+1 to t2-1 do
      st[i-t1] := st[i];
    for i := t2-t1 to dl do
      st[i] := ' ';
  end;
  writeln(st);
end.
```

1. Изначально обнуляем (оба эти значения будут у нас заодно играть роль "флагов") переменные $t1$ и $t2$ для хранения найденных номеров позиций точек и вычисляем длину введенной строки *dl*.

2. В цикле перебираем все символы строки с первого до последнего (с позицией *dl*):

• если текущий (i-й) символ — точка и значение $t1$ все еще равно 0 (т.е. первая точка еще не найдена), то запоминаем найденную позицию точки в первой из двух переменных (в $t1$) и продолжаем цикл, — ведь вторая точка пока еще не найдена;

• если же предыдущее условие не выполнено, то во вложенном операторе **if** проверяем: если текущий символ — точка (и $t1$ не равно нулю, иначе этот случай мы бы "отловили" в предыдущем **if**), то, значит, мы нашли вторую нужную нам точку. Ее позицию мы запоминаем в переменной $t2$ и, раз обе нужные точки найдены, прерываем цикл по команде **break**.

3. Завершив цикл, проверяем: если хотя бы одна из переменных ($t1$ или $t2$) равна нулю, то, значит, цикл завершился сам по себе, а две требуемые точки не найдены, и нужно вместо ответа вывести соответствующее сообщение.

4. Если же и $t1$, и $t2$ не равны нулю (т.е. обе искомые точки в наличии), то нам нужно в строке *st* переписать символы так, как показано на *рис.* 9:

• перебираем символы с позиции $t1 + 1$ до позиции $t2 - 1$ (поскольку сами точки нам не нужны) и перезаписываем их с начала строки: номер позиции, в которую надо записать символ, при этом, очевидно, равен $i - t1$; такую перезапись мы можем осуществить, поскольку при этом каждый нужный символ *сначала* извлекается из строки и только *потом* этот символ может оказаться заменен каким-то другим символом;

• затираем оставшиеся в строке *st* "лишние" символы пробелами, для чего в отдельном цикле перебираем символьные позиции от $t2 - t1$ до конца (до позиции *dl*).

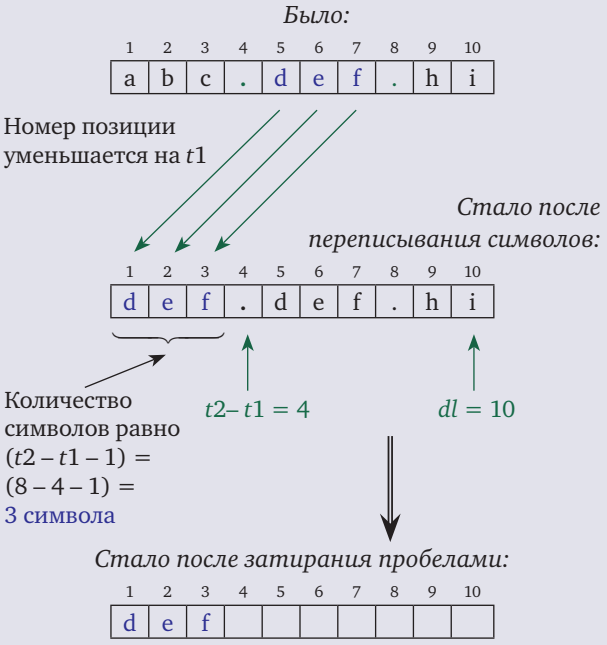


Рис. 9

Решение задачи 15

```
program z_15;
type cifra = set of char;
var st : string;
    dl, i, k, chislo, cif : integer;
    fl : boolean;
    cf : cifra = ['0', '1', '2', '3',
                  '4', '5', '6', '7',
                  '8', '9'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  fl := false;
  for i := 1 to dl do begin
    if (fl = false) and
       (st[i] in cf) then fl := true;
    if (fl = true) and
       not(st[i] in cf) then begin
       fl := false; k := i-1; break;
    end;
  end;
  if fl = true then k := dl;
  if (k <> 1) and (st[k-1] in cf) then
      chislo := StrToInt(st[k-1]) * 10 +
      StrToInt(st[k])
  else
      chislo := StrToInt(st[k]);
```

```
if chislo mod 4 = 0 then
   writeln(st, ' содержит число,
           кратное 4')
else writeln(st, ' не содержит числа,
            кратного 4');
end.
```

Принцип решения во многом совпадает с использованным при решении задачи о числе, кратном 2: здесь в первой части программы тоже определяется позиция последней цифры записи числа. Но дальше начинаются существенные различия.

Согласно признаку деления на 4, нам нужно выделить из записи числа и проверить на кратность четырем не одну, а *две* последние цифры. Поэтому, запомнив в переменной *k* позицию последней цифры числа, мы можем получить число, соответствующее двум последним цифрам исходной числовой записи, так:

chislo := StrToInt(*st*[*k* – 1]) * 10 + StrToInt(*st*[*k*])

(предыдущая цифра, умноженная на 10, плюс последняя цифра).

Но не все так просто! Что если в исходной строке записано число, состоящее всего из одной цифры? Тогда именно ее позиция будет запомнена в переменной *k*, а когда мы попытаемся преобразовать в число предыдущий, нецифровой символ, программа сообщит об ошибке. Поэтому программу придется немного усложнить — добавить проверку этого (*k* – 1)-го символа на принадлежность к множеству символов-цифр. Если это цифра, то требуемое нам двузначное число вычисляется, как указано выше, а иначе мы преобразуем в число только одну *k*-ю цифру:

if *st*[*k* – 1] **in** *cf* **then**
  chislo := StrToInt(*st*[*k* – 1]) * 10 + StrToInt(*st*[*k*])
else *chislo* := StrToInt(*st*[*k*]);

И, наконец, надо учесть случай, когда эта единственная цифра — самая первая в строке (т.е. предыдущий символ вообще не существует и попытка обратиться к (*k* – 1)-му символу приведет к ошибке). Поэтому условие в **if** надо немного дополнить:

if (*k* <> 1) **and** (*st*[*k* – 1] **in** *cf*) **then**
  chislo := StrToInt(*st*[*k* – 1]) * 10 + StrToInt(*st*[*k*])
else *chislo* := StrToInt(*st*[*k*]);

Решение задачи 16

А эту задачу решить гораздо проще, чем предыдущую! Согласно признаку делимости числа на 5, для этого нужно, чтобы последняя цифра была равна 0 или 5. Поэтому достаточно, точно так же, как при решении задачи с числом, кратным 2, найти последний символ-цифру в записи числа в заданной строке, а потом сравнить этот символ с символом '0' и с символом '5' (для чего достаточно логического условия с **or**, — даже дополнительное множество нам не понадобится):

```
program z_16;
type cifra = set of char;
var st : string;
    dl, i, k : integer;
    fl : boolean;
    cf : cifra = ['0', '1', '2', '3',
                  '4', '5', '6', '7',
                  '8', '9'];
begin
  writeln('Введите строку');
  readln (st);
  dl := Length(st);
  fl := false;
  for i := 1 to dl do begin
    if (fl = false) and
       (st[i] in cf) then fl := true;
    if (fl = true) and
        not(st[i] in cf) then
     begin
       fl := false; k := i-1; break;
    end;
  end;
  if fl = true then k := dl;

  if (st[k] = '0') or
     (st[k] = '5') then
      writeln(st, ' содержит число,
                   кратное 5')
  else writeln(st, ' не содержит числа,
                    кратного 5');
end.
```

“О знаках и строках замолвите слово...”

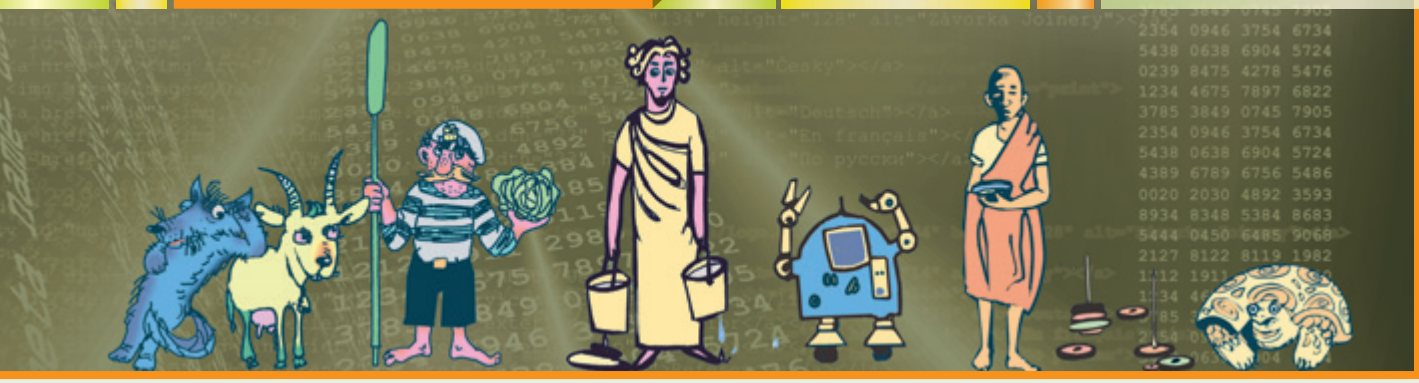
Знаете ли вы...

▶ В некоторых задачах, связанных с обработкой строк, требуется выделить текстовую запись цифры (т.е. символ) и преобразовать ее в собственно цифру (в число, соответствующее этой цифре). Один способ такого преобразования мы уже знаем — использование функций преобразования строковой записи числа в числовое значение. А можно ли обойтись без этих функций?

Да, можно! Для этого служит следующий фрагмент программного кода:

```
// s - символ цифры; n - искомое числовое значение цифры
n := ord(s)-ord('0');  // в апострофах - нуль
```

Постарайтесь самостоятельно разобраться и объяснить, почему эта программная строка позволяет преобразовать символ цифры в саму цифру.



СЕМИНАР

Логические и сдвиговые операции

Д.М. Златопольский,
Москва

Вы, конечно, знаете о логических операциях конъюнкции (или логического умножения), дизъюнкции (логического сложения) и других. А известно ли вам, что в компьютере они используются не только при работе со сложными логическими выражениями при формировании запросов к базам данных, в условных операторах в программах, в функции ЕСЛИ в электронной таблице Microsoft Excel и т.п., но и применительно к числам? Выполняются эти операции в процессоре компьютера (поэтому их называют также логическими командами) над числами, представленными в двоичном виде. Рассмотрим те логические команды, которые выполняются над двумя числами (говорят, что у них два операнда):

1) **AND** (русский вариант — **И**);

2) **OR** (**ИЛИ**);

3) **XOR** (от англ. *eXclusive OR* — **исключающее ИЛИ**).

В отличие от арифметических операций над двумя операндами логические команды являются *поразрядными*. Например, при сложении двух двоичных цифр возможен перенос в старший разряд, а при логических операциях все разряды рассматриваются изолированно друг от друга. Разумеется, действия над всеми разрядами выполняются параллельно и одновременно. Описанные операции называют также “битовыми”.

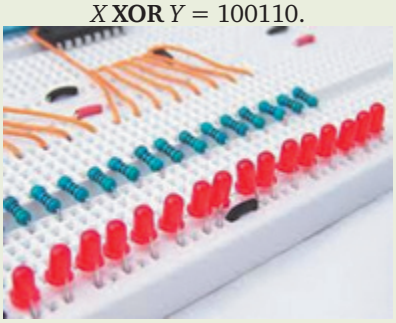
Чтобы было легче понять, в чем заключаются указанные логические операции в процессоре, условимся называть их первый операнд “данными”, а второй — “маской”. Правила выполнения логических операций в каждом разряде представлены в таблице:

X (данные)	*Y* (маска)	*X* AND *Y*	*X* OR *Y*	*X* XOR *Y*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Вам эта таблица ничего не напоминает? Да, конечно, она аналогична таблицам истинности для логических операций над величинами логического типа (с той разницей, что здесь операндами являются двоичные цифры).

Например, при $X = 101011$ и $Y = 1101$ имеем

X **AND** $Y = 1001$,
X **OR** $Y = 101111$,
X **XOR** $Y = 100110$.



Конечно, возможна и поразрядная операция **NOT** (**НЕ**). У нее операнд один. В результате ее выполнения в операнде двоичные цифры 0 и 1 меняются, соответственно, на 1 и 0 (число инвертируется).

Несмотря на то что, как отмечалось, логические операции выполняются в процессоре компьютера, в языке программирования Паскаль имеется возможность дать команду на их выполнение в программе. Для этого в Паскале существуют операции с соответствующими именами: **and**, **or**, **xor**, **not**. Их можно применять к данным целого типа, и результат, который они возвращают, также является целым числом.

Так, результатом операции **and** над двумя операндами является десятичное число, которое соответствует двоичному представлению результата поразрядной операции **И** над исходными операндами. Например, в результате выполнения фрагмента программы:

```
x := 11 and 2;
writeln(x);
```

на экран будет выведено число 2. Обоснование:

	Двоичное представление			
11	1	0	1	1
2	0	0	1	0
11 **and** 2	0	0	1	0
Десятичное представление	2			

Результатом операции **or** над двумя операндами является десятичное число, которое соответствует двоичному представлению результата поразрядной операции **ИЛИ** над исходными операндами. Например, в результате выполнения фрагмента программы:

```
x := 11 or 2;
writeln(x);
```

на экран будет выведено число 11. Обоснование:

	Двоичное представление			
11	1	0	1	1
2	0	0	1	0
11 **or** 2	1	0	1	1
Десятичное представление	11			

Результатом операции **xor** над двумя операндами является десятичное число, которое соответствует двоичному представлению результата поразрядной операции **ИЛИ** над исходными операндами. Например, в результате выполнения фрагмента программы:

```
x := 11 xor 2;
writeln(x);
```

на экран будет выведено число 9. Обоснование:

	Двоичное представление			
11	1	0	1	1
2	0	0	1	0
11 **xor** 2	1	0	0	1
Десятичное представление	9			

Результатом операции **not** с одним операндом является десятичное число, которое соответствует двоичному представлению результата поразрядной операции **НЕ** над исходным операндом. Например, в результате выполнения фрагмента программы:

```
Var a: byte;
...
a := 11;
writeln(not a);
```

на экран будет выведено число 244. Обоснование:

	Двоичное представление							
11	0	0	0	0	1	0	1	1
not 11	1	1	1	1	0	1	0	0
Десятичное представление	244							

Над числами *в регистрах* процессора можно выполнять также две так называемые “сдвиговые” команды:

1) **SHL** (от **Sh**ift **L**eft — сдвинуть влево);
2) **SHR** (от **Sh**ift **R**ight — сдвинуть вправо).

В результате выполнения первой в регистрах процессора происходит поразрядный сдвиг двоичных цифр на заданное количество разрядов влево. При этом соответствующее число цифр в старших разрядах теряется, а освободившиеся младшие разряды заполняются нулями. Например, если в восьми разрядах находилось двоичное число:

1	1	0	0	1	0	0	1

то после сдвига влево на два разряда оно примет вид:

0	0	1	0	0	1	0	0

В результате выполнения команды **SHR** в регистрах процессора происходит поразрядный сдвиг двоичных цифр на заданное количество разрядов вправо. При этом соответствующее число цифр в младших разрядах теряется, а освободившиеся старшие разряды заполняются нулями. Например, если в восьми разрядах было представлено двоичное число:

1	1	0	0	1	0	0	1

то после сдвига вправо на два разряда оно примет вид:

0	0	1	1	0	0	1	0

В языке Паскаль также имеется возможность выполнить эти команды в программе.

Операция `shl` имеет формат:

```
<a> shl <b>
```

— где `<a>` и `<b>` — натуральные числа.

Результатом ее выполнения будет десятичное число, которое соответствует результату поразрядного сдвига влево на *b* разрядов двоичного представления операнда *a*.

Аналогично, результатом выполнения операции `shr`:

```
x := a shr b
```

будет десятичное число *x*, которое соответствует результату поразрядного сдвига вправо на *b* разрядов двоичного представления операнда *a*.

Возможен также циклический сдвиг влево, при котором “уходящие” один или несколько старших разрядов “появляются” на месте самого младшего. Например, если в восьми разрядах находилось двоичное число:

1	0	0	0	1	0	0	1

то после циклического сдвига влево на два разряда оно примет вид:

0	0	1	0	0	1	1	0

При циклическом сдвиге вправо “уходящие” младшие разряды “появляются” на месте старших. Например, если в восьми разрядах находилось двоичное число:

1	1	0	0	1	0	0	1

то после циклического сдвига вправо на два разряда оно примет вид:

0	1	1	1	0	0	1	0

Команды для выполнения циклических сдвигов в программе на языке Паскаль отсутствуют.

Задания для самостоятельной работы

1. Рассчитайте значения:
а) 1101101 AND 10101;
б) 1011011 OR 11001;
в) 1001101 XOR 10111.

2. Определите:

1) чему равен результат операции **И** при нулевой маске;

2) чему равен результат операции **ИЛИ** при маске:

а) из всех единиц;

б) в виде нуля;

3) какая операция и с какой маской позволяет выделить (получить) младший разряд двоичного числа;

4) какой получится результат, если к какому-то числу дважды применить операцию исключающего **ИЛИ**;

5) какой получится результат, если операцию исключающего **ИЛИ** применить к двум одинаковым числам;

6) какой получится результат, если к какому-то числу применить операцию исключающего **ИЛИ** с маской из всех единиц.

3. Выясните, можно ли определить десятичный результат выполнения поразрядной логической операции **NOT** с целым десятичным числом типа *byte* "в уме". Почему?

4. Установите, можно ли применять поразрядные логические операции к одному или двум отрицательным операндам.

5. Определите результат выполнения следующего фрагмента программы:

```
var a: byte; b: shortint;
BEGIN
  a := 11;
  writeln(not a);
  b := 11;
  writeln(not b)
END.
```

Почему выводятся разные значения?

6. Объясните, почему a **and** $-(a + 1) = 0$. (Например, 51 **and** $-52 = 0$.)

7. Установите, как изменится число a в результате выполнения оператора присваивания

```
a := a shr 1;
```

1) при четном a;

2) при нечетном a.

Как можно обобщить оба ответа?

8. Определите, как изменится число a, равное 1024, в результате выполнения оператора присваивания:

```
a := a shr k.
```

9. Можно ли ответы по двум предыдущим заданиям распространить на поразрядный сдвиг влево? Можно ли утверждать, что при таком сдвиге число a увеличится? А что уменьшится?

10. Установите, можно ли применять сдвиговые операции к отрицательным числам.

11. Определите, как изменится число a при циклическом сдвиге вправо на один разряд, в случае, когда в крайнем правом разряде:

1) записан 0;

2) записана 1.

12. Установите формулу, по которой можно определить, как изменится число a при циклическом сдвиге вправо на b разрядов, в случае, когда в b правых разрядах:

1) записаны нули;

2) записаны единицы.

Что можно сказать об изменении значения a в этих случаях (оно увеличится или т.п.)?

13. Определите, как изменится число a при циклическом сдвиге влево на один разряд, в случае, когда в крайнем левом 8-м разряде:

1) записан 0;

2) записана 1.

Можно ли сделать вывод об изменении значения a в этих случаях (оно увеличится или т.п.)?

14. Установите формулу, по которой можно определить, как изменится число a при циклическом сдвиге влево на b разрядов, в случае, когда в b левых разрядах:

1) записаны нули;

2) записаны единицы.

В обоих случаях принять, что число a записано в восьми разрядах. Можно ли сделать вывод об изменении значения a в этих случаях (оно увеличится или т.п.)?

15. В восьми двоичных разрядах записано число n. Как получить:

1) цифру в первом (крайнем справа) разряде;

2) цифру в четвертом разряде;

3) цифру в k-м разряде?

Примеры использования логических команд в компьютере приведены в очень интересной и полезной книге [1]. В одном из будущих номеров "В мир информатики" будет также рассмотрен ряд задач с использованием поразрядных логических и сдвиговых операций.

Литература

1. *Еремин Е.А.* Популярные лекции по устройству компьютера. СПб.: БХВ-Петербург, 2003.

"ЛОМАЕМ" ГОЛОВУ

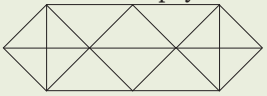
Гастролер

Однажды некий артист-гастролер первый вторник месяца провел в Иркутске, а первый вторник после первого понедельника — в Новосибирске. В следующем месяце он первый вторник провел в Воронеже, а первый понедельник — в Москве.

Какого числа и какого месяца он был в каждом из указанных городов?

Сколько треугольников на рисунке?

Подсчитайте, пожалуйста, количество треугольников на рисунке.



Ответы присылайте в редакцию.

ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным ранее

1. Пять ребусов на одну тему

Ответы

Ребус № 1: ИСПОЛНИТЕЛЬ.
Ребус № 2: ЧЕРТЕЖНИК.
Ребус № 3: РОБОТ.
Ребус № 4: ЧЕРЕПАШКА.
Ребус № 5: КОМАНДА.

Правильные ответы прислали:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Бибичкова Екатерина и Костина Нина, Совхозная средняя школа, Московская обл., Серебряно-Прудский р-н, поселок Успенский, учитель **Жарикова Е.Н.**;

— Гунбина Алина (в прошлом учебном году ученица 2-го класса), средняя школа села Новотроицкое, Кировская обл., Шабалинский р-н, учитель **Гунбина И.В.**;

— Елфимова Анна, Республика Коми, г. Сыктывкар, школа № 18, учитель **Гладких Ю.В.**;

— Костюнин Александр, Лешукова Мария и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Неофитова Елена, средняя школа села Янтиково, Чувашская Республика, учитель **Неофитова Н.Н.**;

— Серебряков Евгений и Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Семин Влад, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Товмасян Арсен, средняя школа села Урман, Республика Башкортостан, Иглинский р-н, учитель **Товмасян М.Г.**;

— Хатанзейская Кристина, основная школа поселка Каратайка, Архангельская обл., учитель **Безумова В.А.**;

— Шадрина Юлия, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Воеводина Р.В.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

Большинство приславших ответы правильно указали, что ребусы связаны с темой “Исполнители алгоритмов”.

2. Числовой ребус “Одни звездочки”

Ответ

В ребусе зашифрован пример деления “столбиком” числа 1 на 8.

Правильный ответ представили:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Васинская Екатерина и Сафиуллин Ильдар, Республика Башкортостан, г. Стерлитамак, школа № 17, учитель **Орлова Е.В.**;

— Краснёнкова Ксения, г. Благовещенск, школа № 11, учитель **Вернер Ю.Ю.**;

— Товмасян Арсен, средняя школа села Урман, Республика Башкортостан, Иглинский р-н, учитель **Товмасян М.Г.**;

— Хатанзейская Кристина, основная школа поселка Каратайка, Архангельская обл., учитель **Безумова В.А.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

3. Задача “Стоимость работы”

Напомним условие.

Когда-то один человек собирался построить дом и выяснил, что ему придется заплатить:

— 1100 рублей обойщику и маляру;
— 1700 рублей маляру и жестянщику;
— 1100 рублей жестянщику и электрику;
— 3300 рублей электрику и плотнику;
— 5300 рублей плотнику и каменщику;
— 3200 рублей каменщику и маляру.

Сколько запросил каждый из мастеров за свою работу?

Правильные ответы представили:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Ворожейкина Ксения и Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Григоренко Дмитрий, Есипова Мария, Круглякова Мария и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Краснёнкова Ксения, г. Благовещенск, школа № 11, учитель **Вернер Ю.Ю.**;

— Неофитова Елена, средняя школа села Янтиково, Чувашская Республика, учитель **Неофитова Н.Н.**;

— Серебряков Евгений и Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Товмасян Арсен, средняя школа села Урман, Республика Башкортостан, Иглинский р-н, учитель **Товмасян М.Г.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

Решение

Самое краткое решение привели Карина Галушкова и Арсен Товмасян. Карина предложила выразить стоимость работы разных мастеров через стоимость работы маляра (который фигурирует в условии чаще других):

Обойщик = 1100 – *Маляр* (1)

Жестянщик = 1700 – *Маляр* (2)

Каменщик = 3200 – *Маляр* (3)

Электрик = 1100 – *Жестянщик* = 1100 – (1700 – – *Маляр*) = *Маляр* – 600 (4)

Плотник = 3300 – *Электрик* = 3300 – (*Маляр* – – 600) = 3900 – *Маляр* (5)

Подставив затем в уравнение *Плотник* + *Каменщик* = 5300 информацию из (5) и (2), получим:

3900 – *Маляр* + 3200 – *Маляр* = 5300, откуда

Маляр = 900,

Обойщик =1100 – 900 = 200,

Жестянщик = 1700 – 900 = 800,

Каменщик = 3200 – 900 = 2300,

Электрик = 900 – 600 = 300,

Плотник = 3900 – 900 = 3000.

Арсен решил задачу, выразив стоимость работы разных мастеров через стоимость работы обойщика (которую он обозначил X).

Тогда маляр запросил (1100 – X) руб.,

жестянщик — 1700 – (1100 – X) = (600 + X) руб.,

электрик — 1100 – (600 + X) = (500 – X) руб.,

плотник — 3300 – (500 – X) = (2800 + X) руб.,

каменщик — 5300 – (2800 + X) = (2500 – X) руб.

Так как каменщик и маляр запросили вместе 3200 руб., то можем записать:

(2500 – X) + (1100 – X) = 3200,

откуда X = 200 руб.

4. Задача “Кто съел варенье?”

Напомним, что необходимо было определить, кто съел варенье — Петя, Вася или Маша, если известно, что в своих ответах:

— Петя: “Я не ел. Маша тоже не ела”;

— Вася: “Маша действительно не ела. Это сделал Петя”;

— Маша: “Вася врет. Это он съел”,

двое оба раза сказали правду, а третий один раз сказал правду и один раз соврал.

Решение

Метод допущений можно не применять. Ясно, что одновременно сказать дважды правду не могли:

1) Петя и Вася;

2) Вася и Маша.

Остается, что правду оба раза сказали Маша и Петя, то есть съел варенье Вася.

Правильные ответы представили:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Костюнин Александр, Лешукова Мария и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Семин Влад, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Товмасян Арсен, средняя школа села Урман, Республика Башкортостан, Иглинский р-н, учитель **Товмасян М.Г.**;

— Неофитова Елена, средняя школа села Янтиково, Чувашская Республика, учитель **Неофитова Н.Н.**;

— Хатанзейская Кристина, основная школа поселка Каратайка, Архангельская обл., учитель **Безумова В.А.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

Отметим ответ Арсена Товмасяна, решившего задачу методом схем.

5. Софизм “4 = 8”

Объяснение софизма привели:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Серебряков Евгений и Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**;

— Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**

6. Фокус “Угадывание числа”

Напомним, что необходимо было с помощью формул доказать, что если кто-то задумал число, умножил его само на себя, к полученному результату прибавил удвоенное задуманное число и еще 1, то по объявленному результату можно назвать задуманное число следующим образом: надо из объявленного числа извлечь квадратный корень, а затем вычесть число 1.

Ответы представили:

— Базылев Юрий, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Ворожейкина Ксения и Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

7. “Шесть вопросов. Вариант 5”

Ответы:

1. Созвездие Рыбы находится между зодиакальными созвездиями Водолей и Овен.

2. Чемпионом СССР по футболу в 1961 году в классе “А” стала команда киевского “Динамо”.

3. Русалка изображена на гербе Варшавы, столицы Польши.

4. Лауреатом Нобелевской премии мира за 1976 год стали Бетти Уильямс (*Betty Williams*) и Мейрид Корриган (*Mairead Corrigan*), Ирландия.

5. Название своей столицы китайцы произносят как “Бэйдзин”.

6. Денежной единице Веймарской республики, для того чтобы отличить ее от полновесных довоенных золотых, имевших хождение до августа 1914 года, было дано название “Бумажная марка”.

По перечисленным ответам у читателей мнение единодушное. А вот по вопросу о том, кто был тренером киевского “Динамо” — чемпиона СССР по футболу в 1961 году в классе “А” — мнения разделились. Указаны: Олег Ошенков, Виктор Маслов, Вячеслав Соловьев, Николай Горшков и даже Валерий Газаев, родившийся в 1954 (!) году. Правильный ответ — Вячеслав Соловьев.

Ответы представили:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Бибичкова Екатерина и Костина Нина, Совхозная средняя школа, Московская обл., Серебряно-Прудский р-н, поселок Успенский, учитель **Жарикова Е.Н.**;

— Буханов Василий, Григорьев Кирилл и Юхтенко Илья, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Зорихин Алексей, Свердловская обл., г. Нижняя Салда, школа № 7, учитель **Зорихина Н.Ю.**;

— Костюнин Александр, Лешукова Мария и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Микулик Илья, г. Астрахань, школа № 33 им. Н.А. Мордовиной, учитель **Лепехина С.М.**;

— Шадрина Юлия, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Воеводина Р.В.**;

— Шутихина Ксения, Республика Башкортостан, г. Уфа, лицей № 60, учитель **Гильзер Н.В.**

За активное участие в выполнении заданий рубрики “Поиск информации” ряд перечисленных читателей награжден дипломами (список награжденных приведен в предыдущем выпуске).

8. Статья “В казино”

Напомним, что в статье рассматривались задачи, связанные с игрой в “рулетку”.

Ответ на задание 1

Количество выигрышных номеров при общем числе номеров 10 000 равно 746 (выигрышный номер — тот, который делится на целую часть своего кубического корня).

Задание 2 было связано с разработкой программы для подсчета количества выигрышных номеров при заданном общем числе номеров N на колесе рулетки. Подсчет должен проводиться при рассмотрении не всех чисел, а их кубических корней, точнее — целой части последних. Например, при 1000 чисел целая часть корня (назовем эту величину — *корень*) может быть равна 1, 2, …, 10. Если *корень* = 1, то ему соответствуют выигрышные номера от 1 до 7, ибо для каждого из них $[\sqrt[3]{n}] = 1$ (квадратные скобки означают целую часть числа, записанного между ними). При *корень* = 2 выигрышными являются только четные числа из диапазона от $2^3 = 8$ до $3^3 - 1 = 26$. Среди номеров от $3^3 = 27$ до $4^3 - 1 = 63$ такие номера — только те, которые делятся на 3 (*корень* = 3) и т.д.

Диапазоны чисел, которые и выигрышные номера, соответствующие значению величины *корень* в рассматриваемом случае, приведены в таблице:

Корень	Диапазон чисел	Из них выигрышные номера
1	1 – 7, или $1^3 – (2^3 – 1)$	Все
2	8 – 26, или $2^3 – (3^3 – 1)$	Четные
3	27 – 63, или $3^3 – (4^3 – 1)$	Кратные трем
4		Кратные четырем
…		
10	1000 (в рассматриваемом случае)	1000

Из таблицы следует, что при фиксированном значении величины *корень* выигрышные номера равны *корень*^{3}, *корень*^{3} + *корень*, *корень*^{3} + 2*корень*, …, (*корень* + 1)^3 – 1. С учетом этого программа для нахождения количества выигрышных номеров в общем случае имеет вид:

```
алг В_казино
нач цел N, макс_корень, корень, число,
        кол-во_выигрышей
 вывод нс, "Задайте общее число номеров"
 ввод N
 макс_корень := цел(N ** (1/3))
 число := 0
 кол-во_выигрышей := 0
 нц для корень от 1 до макс_корень – 1
  нц пока число <= (корень + 1) ** 3 – 1
   |Очередное выигрышное число
   число := число + корень
```

```
    кол-во_выигрышей := кол-во_выигрышей + 1
  кц
  |Возвращаемся к числу — началу след. группы
  число := число — корень + 1
кц
|Последняя группа может быть неполной
нц пока число <= N
  число := число + макс_корень
  кол-во_выигрышей := кол-во_выигрышей + 1
кц
|Так как учтено число (N + макс_корень),
|то уточняем кол-во выигрышных номеров
кол-во_выигрышей := кол-во_выигрышей — 1
вывод нс, "Кол-во выигрышных номеров равно "
вывод количество_выигрышей
кон
```

Ответы и программы прислали:

— Базылев Юрий, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Востриков Игорь, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Журавлев Алексей, Кондаков Роман, Пехов Андрей и Скобелев Николай, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Семеняченко Кирилл, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Хотеев Сергей, Москва, гимназия № 1530, учитель **Шамшев М.В.**

Все перечисленные читатели будут награждены дипломами. Поздравляем!

Три одноклассника

Три одноклассника — Влад, Тимур и Юра — встретились спустя 10 лет после окончания школы. Выяснилось, что один из них стал врачом, второй — 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. Интересно, что 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.

Шесть карточек

Для составления цепочек букв разрешается использовать 6 карточек с буквами *А*, *Б*, *Е*, *Ж*, *И*, *К*. Каждая цепочка должна состоять из всех шести карточек, при этом должны соблюдаться правила:

1) любая цепочка начинается гласной буквой;

2) после гласной буквы не может снова идти гласная, а после согласной — согласная;

3) буквы в цепочке не должны повторяться.

Сколько всего существует таких цепочек?

Бидоны с молоком

Надоенное на ферме молоко заполняет несколько 50-литровых бидонов. Если его разлить в 40-литровые бидоны, то понадобится на 5 бидонов больше, и один из них останется неполным. Если же это молоко разлить в 70-литровые бидоны, то понадобится на 4 бидона меньше, и один из них тоже останется неполным. Сколько 50-литровых бидонов заполняет надоенное молоко?

Задачу желательно решить, используя электронную таблицу Microsoft Excel (допускается и аналитическое решение).

Проложить маршрут

Имеется шахматная доска с обозначением клеток согласно стандартной шахматной нотации ($a1$ — нижняя левая, ..., $h8$ — верхняя правая):

$a8$	$b8$	$c8$	$d8$	$e8$	$f8$	$g8$	$h8$
$a7$	$b7$	$c7$	$d7$	$e7$	$f7$	$g7$	$h7$
$a6$	$b6$	$c6$	$d6$	$e6$	$f6$	$g6$	$h6$
$a5$	$b5$	$c5$	$d5$	$e5$	$f5$	$g5$	$h5$
$a4$	$b4$	$c4$	$d4$	$e4$	$f4$	$g4$	$h4$
$a3$	$b3$	$c3$	$d3$	$e3$	$f3$	$g3$	$h3$
$a2$	$b2$	$c2$	$d2$	$e2$	$f2$	$g2$	$h2$
$a1$	$b1$	$c1$	$d1$	$e1$	$f1$	$g1$	$h1$

Из некоторой начальной клетки необходимо проложить маршрут в клетку $a1$, соблюдая следующее правило: каждый ход делается либо на одну клетку влево, либо на одну клетку вниз, либо на одну клетку вниз и на одну клетку влево. Например, из клетки $d3$ допустимы ходы на клетки $c3$, $d2$, $c2$.

Перечислите все допустимые маршруты, ведущие из начальной клетки $c3$. Для сокращения записи примите следующую систему обозначения ходов:

Л — ход влево;

Д — ход по диагонали (“вниз и влево”);

Н — ход вниз.

Каждый маршрут запишите в виде набора букв, которые соответствуют обозначениям ходов.

Наша любимая двойка ☺

Используя пять раз цифру 2, знаки арифметических действий и скобки, запишите выражение, значение которого будет равно:

1) 11; 2) 15; 3) 12 321.

Примечание. Надеемся, что все читатели раздела, в названии которого фигурирует слово “информатика”, поняли, какую двойку мы имеем в виду.

Восстановить пример

Замените звездочки недостающими цифрами, чтобы пример на умножение был верен.

Любой звездочкой может быть любая цифра.

```
      *  *  7
  ×      3  *  *
---------------
      *  0  *  3
      *  1  *
   *  5  *
---------------
   *  7  *  *  3
```

“ЖЕЛЕЗО”

Еще два вентиля

В статье [1] были описаны так называемые “логические вентили” — технические устройства, с помощью которых можно реализовать логические операции конъюнкции (вентиль “И”) и дизъюнкции (вентиль “ИЛИ”). Используя их, можно сконструировать двоичный сумматор — устройство для сложения двух двоичных чисел. Но для этого понадобятся еще два логических вентиля.

В обоих применяется выход реле, в котором есть напряжение, когда реле не сработало (то есть тот выход, что используется в инверторе, рассмотренном в [1]). Например, в следующей конфигурации выход одного реле подает питание на вход второго реле (*рис.* 1).

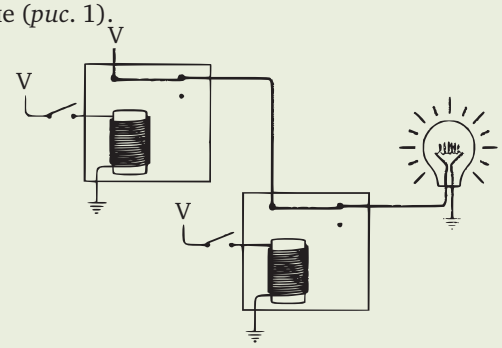


Рис. 1

Лампочка горит, когда оба входа отключены от питания.

Если верхний переключатель замкнут, лампочка гаснет (*рис.* 2).

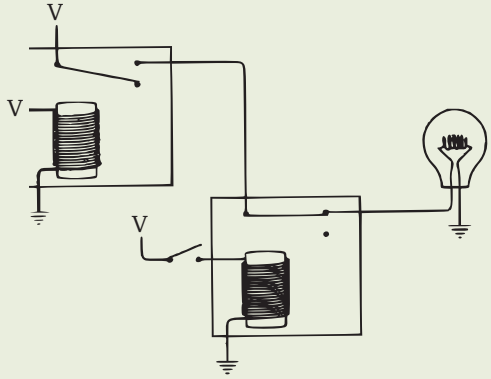


Рис. 2

Это происходит потому, что питание перестает поступать на второе реле.

Лампочка также не горит, если замкнут нижний переключатель (*рис.* 3).

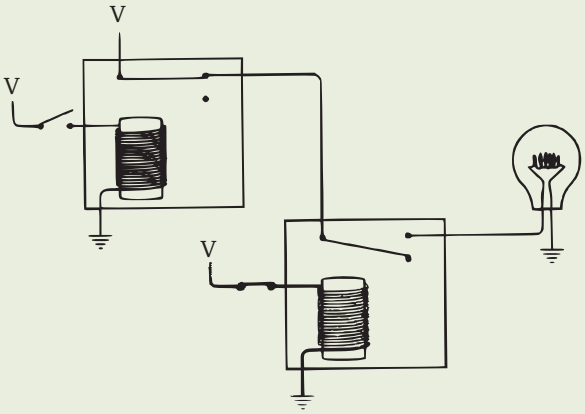


Рис. 3

Если оба переключателя замкнуть, лампочка также не горит (*рис.* 4).

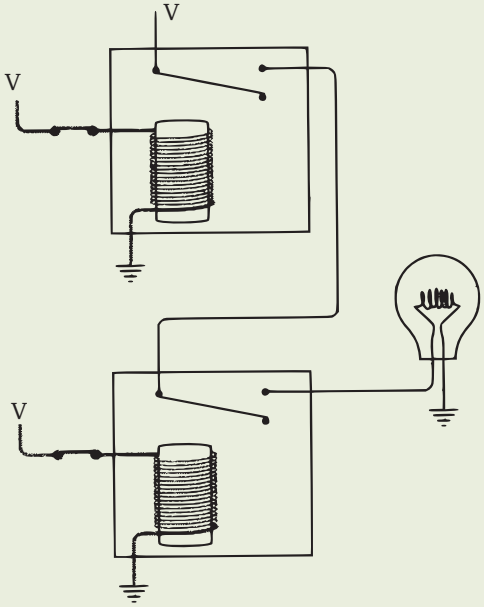


Рис. 4

Итак, можно сказать, что выходной сигнал равен 1, только если на оба входа подается 0, и равен 0 во всех других случаях. Такое поведение в точности противоположно поведению вентиля “ИЛИ”. Поэтому такая схема называется “вентилем ИЛИ–НЕ”. Его обозначение:

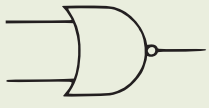


Рис. 5

Вентиль ИЛИ–НЕ эквивалентен схеме:

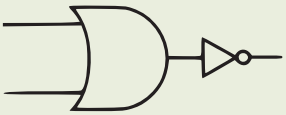


Рис. 6

Работу вентиля “ИЛИ–НЕ” иллюстрирует следующая таблица:

ИЛИ–НЕ	0	1
0	1	0
1	0	0

Вот еще один способ соединения двух реле:

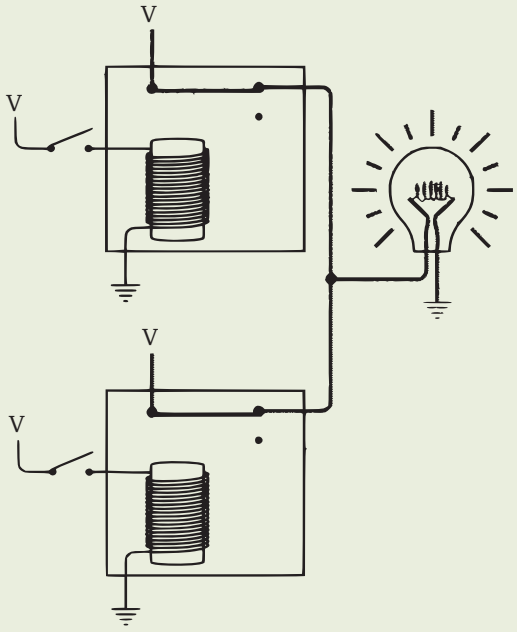


Рис. 7

Здесь оба выхода соединены друг с другом, как в вентиле “ИЛИ”, но с использованием других контактов. Лампочка не погаснет, если замкнуть верхний переключатель (*рис.* 8).

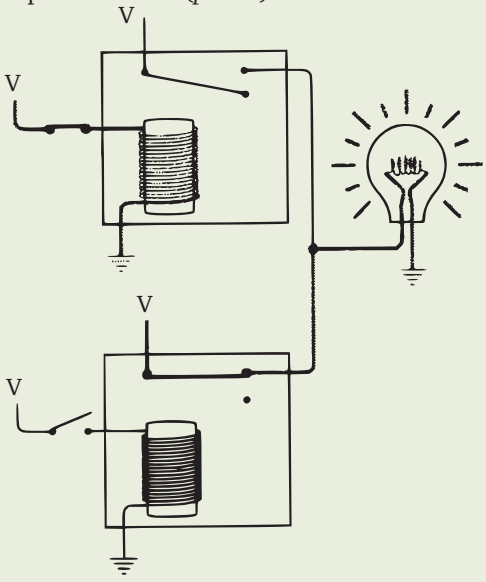


Рис. 8

Не погаснет она, если будет замкнут только нижний переключатель (*рис.* 9).

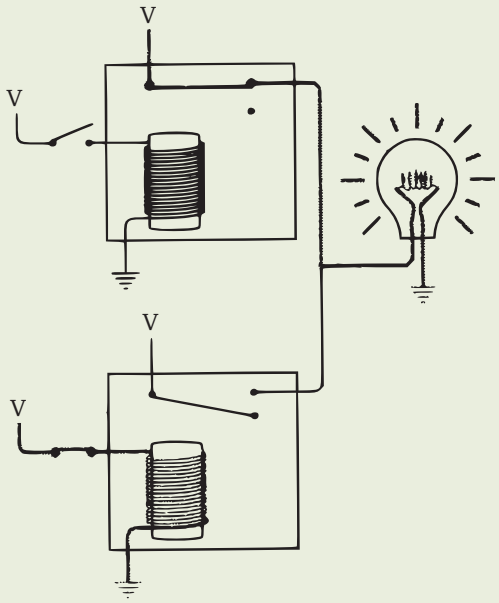


Рис. 9

Только если оба переключателя замкнуты, лампочка гаснет (*рис.* 10).

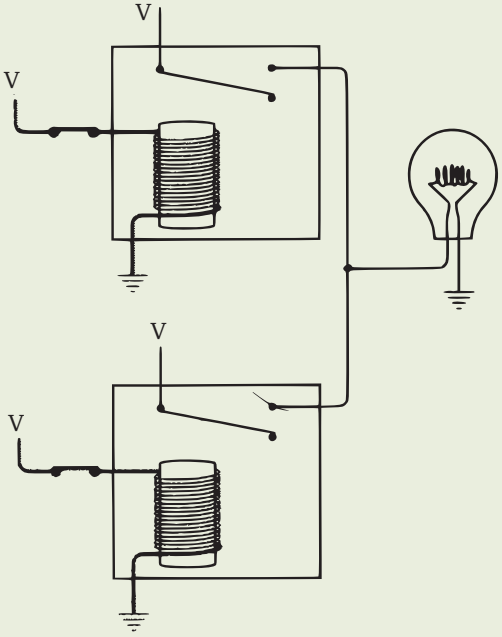


Рис. 10

Можно сказать, что выходной сигнал равен 0, только если на оба входа подается 1, и равен 1 во всех других случаях. Такое поведение в точности противоположно поведению вентиля “И”. Поэтому такая схема называется “вентилем И–НЕ”. Его обозначение

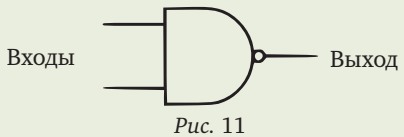


Рис. 11

Работу вентиля “И–НЕ” иллюстрирует следующая таблица:

И–НЕ	0	1
0	1	1
1	1	0

Итак, мы рассмотрели четыре различных способа соединения реле с двумя входами и одним выходом. Чтобы каждый раз не рисовать реле, мы назвали эти схемы “логическими вентилями” и решили использовать для них стандартные обозначения:

1) вентиль “И”:

Рис. 12

2) вентиль “ИЛИ”:

Рис. 13

3) вентиль “НЕ–ИЛИ” — см. *рис.* 5;

4) вентиль “НЕ–И” — см. *рис.* 11.

Зависимость выходного сигнала каждого из четырех вентилей от сигналов на входе иллюстрируется следующей таблицей:

И	**0**	**1**
0	0	0
1	0	1
ИЛИ	**0**	**1**
0	0	1
1	1	1
ИЛИ–НЕ	**0**	**1**
0	1	0
1	0	0
И–НЕ	**0**	**1**
0	1	1
1	1	0

О том, как, используя эти четыре логических вентиля, сконструировать двоичный сумматор, будет рассказано в следующем выпуске.

Литература

1. Выбрать кошку помогают… реле. / “В мир информатики” № 169 (“Информатика” № 15/2011).

2. От кошек — через переключатели — к компьютерам. / “В мир информатики” № 168 (“Информатика” № 14/2011).

3. *Петцольд Ч.* Код. М.: Издательско-торговый дом “Русская редакция”, 2001.

ИСТОРИЯ ИНФОРМАТИКИ

4 декабря — День российской информатики

Александр Нитусов

Прогресс информатики сравним со взрывом. Приручение домашних животных растянулось на сотни или даже тысячи лет, корабли и кареты проделали ненамного меньший путь, чтобы стать массовым транспортом, а компьютеры вошли в общее употребление буквально за пару десятилетий. Лет 70 назад их попросту не было.

Праздников — много. Кроме Нового года, Дня Победы и других, популярны профессиональные “дни”: День учителя, День рыбака и т.д. В основе любого праздника, официального, “спонтанного”, общего или местного, всегда лежит всплеск чувств по поводу важного события.

На заре человечества было просто: убили мамонта и все племя сыто — ура, веселье! Однако съели и забыли. Это еще не праздник, а “празднование” — эпизод. Со временем человек стал отмечать регулярные события — природные циклы. От них жизнь зависит, они ее регламентируют — заодно и календарь (а подсчет дней, месяцев — это уже зачатки математики — “прародительницы” информатики). Приход весны — конец зимнего холода (и голода), дошел до нас как праздник Пасхи, сбор урожая — День урожая во многих странах отмечают до сих пор и т.д.

Потом пришел черед праздновать события “разовые”, но этапные, изменявшие жизнь. Скажем, день постройки главного храма, особо важного корабля и, уж само собой, рождения нового фараона ☺.

Ну а “чей-то День” — это дань уважения и благодарности тем, кто не эпизодически, а постоянно трудится, чтобы наше общество существовало и развивалось. Хотя эти “дни” тоже приурочивают к каким-то конкретным событиям.

Чем важнее событие, тем дольше существует праздник. Главное его свойство — он появляется “сам” (а “утвердят” — это потом).

Так, в 1960–70-е гг. в подмосковном Зеленограде работал замечательный ученый и человек, конструктор ЭВМ специального назначения Давлет Исламович Юдицкий. Самого Юдицкого давно уже нет, а друзья и коллеги продолжают отмечать день его рождения. Это их праздник.

Недавно появился и *День российской информатики*. Он отмечается 4 декабря. Вряд ли кто-то сомневается, что информатика преобразила нашу жизнь, но вот от чего отсчитывать ее историю? Когда праздник праздновать? От дня начала производства механического вычислителя — арифмометра Однера? А ведь счетные приборы на Руси были и ранее. Но и точная дата появления русских счетов неизвестна… Так почему же все-таки 4 декабря?

Днем рождения отечественной информатики признано 4 декабря 1948 года — день регистрации патента[1] № 10475 на автоматическую цифровую вычислительную машину, выданного члену-корреспонденту Академии наук СССР Исааку Семеновичу Бруку и инженеру Баширу Искандаровичу Рамееву Государственным комитетом Совета Министров СССР по внедрению передовой техники в народное хозяйство. Это был первый советский патент на цифровую ЭВМ, первый шаг на пути “наших компьютеров”.

[1] В СССР роль патентов выполняли государственные авторские свидетельства на изобретение. — **Прим. ред.**

Исаак Семенович Брук

Несмотря на большую разницу в возрасте и ученых званиях, авторы разрабатывали проект почти как равные коллеги. Их жизненные пути очень несхожи, но к этапному изобретению привело умение "чувствовать пульс науки" и огромный творческий труд.

Исаак Семенович Брук (1902–1974) родился в Минске в бедной семье мелкого служащего. В 1925 г. он защитил диплом на кафедре электротехники в Московском высшем техническом училище, работал во Всесоюзном электротехническом институте (ВЭИ) — головном исследовательском центре по электроэнергетике, а в 1935 г. "обосновался" в Энергетическом институте Академии наук (ЭНИН АН СССР), где организовал лабораторию сетей электроснабжения.

Энергоснабжение, особо важное при бурном росте промышленности, требовало огромного объема расчетов. В 1935 г. Брук сделал аналоговую вычислительную машину — "электрический стол переменного тока" для моделирования и расчета сетей электропередачи (хранится в Политехническом музее).

В 1936 г. он успешно защитил докторскую диссертацию и занялся огромным механическим вычислителем — интегратором, похожим на гипертрофированную машину Бэббиджа (такие монстры разрабатывали и в Англии, и в США). Но, хотя он и решал дифференциальные уравнения до 6-го порядка, эпоха механических вычислений закончилась. Впрочем, сам проект был удачным, и Брука избрали в Академию наук.

Во время войны он конструировал механические аналоговые устройства управления авиационными пушками, придумал синхронизатор для стрельбы сквозь вращающийся пропеллер и продолжал работы по энергосистемам.

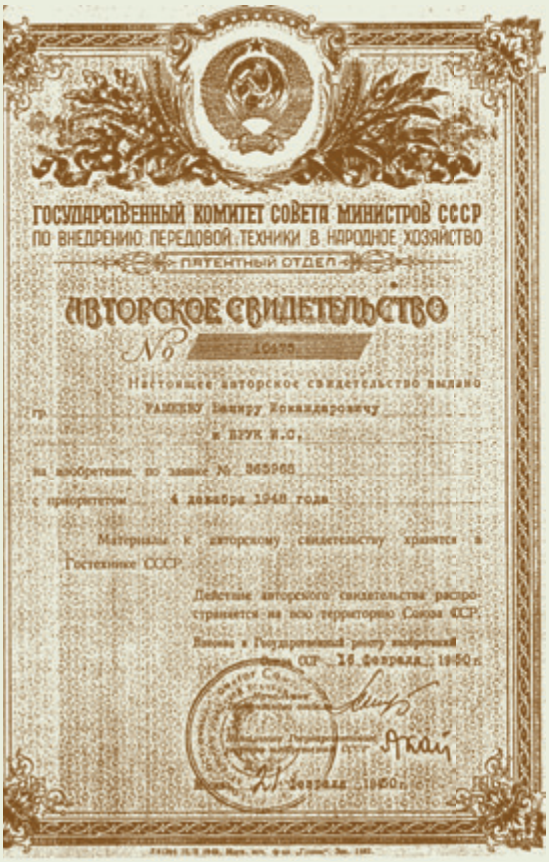
В 1947 г. И.С. Брука избрали в популярную тогда Академию артиллерийских наук. В первые послевоенные годы его лаборатория сконструировала электронный дифференциальный анализатор (ЭДА), решавший интегральные уравнения до 20-го порядка. ЭДА — полноценная аналоговая электронная вычислительная машина.

В годы Второй мировой войны на Западе уже появлялись отдельные электрические вычислительные устройства. Еще в 1940–41 гг. Сергей Алексеевич Лебедев — один из мировых лидеров ЭВМ и информатики — говорил, что полноценная вычислительная машина должна быть только электронной, цифровой и с двоичной арифметикой.

В общем, "цифровой компьютер витал в воздухе". Брук много лет шел к ЭВМ, и нетрудно представить его радость, когда в мае 1948 г. создатель и начальник института ЦНИИ № 108 (радиолокация и связь) академик и адмирал Аксель Иванович Берг (проницательный ученый и гениальный организатор, создатель советской электронной промышленности) направил к нему молодого специалиста, своего ученика Башира Рамеева, интересовавшегося электронными вычислителями.

Башир Искандарович Рамеев

Башир Искандарович Рамеев, внук Закира Рамеева — члена Российской Государственной думы, золотопромышленника и татарского поэта, родился 1 мая 1918 г. в местечке Баймак. Его мать Шарипзада Рамеева — потомок старинного дворянского рода Дашковых. Отец — горный инженер Искандар Рамеев, окончил в 1914 г. Горную академию во Фрайберге (Германия), успешно работал, но в 1938 г. был репрессирован и в 1943 г. умер.



ГОСУДАРСТВЕННЫЙ КОМИТЕТ СОВЕТА МИНИСТРОВ СССР
ПО ВНЕДРЕНИЮ ПЕРЕДОВОЙ ТЕХНИКИ В НАРОДНОЕ ХОЗЯЙСТВО
ПАТЕНТНЫЙ ОТДЕЛ

АВТОРСКОЕ СВИДЕТЕЛЬСТВО

№ 104175

Настоящее авторское свидетельство выдано
РАМЕЕВУ Баширу Искандаровичу
и БРУК И.С.

на изобретение по заявке № 365965
с приоритетом 4 декабря 1948 года

Материалы к авторскому свидетельству хранятся в Госпатенте СССР

Действие авторского свидетельства распространяется на всю территорию Союза ССР

Башира, учившегося в Московском энергетическом институте, исключили как "сына врага народа", но всю последующую жизнь он в любых условиях занимался самообразованием. В 1938 г. с трудом устроился на "Башрадио", сам изучал основы радиотехники и смежные науки, а в 1939 г. благодаря знаниям радио (тогда оно было необычайно популярным) и талантам изобретателя стал техником в московском ЦНИИ связи.

Еще школьником он построил и послал в Москву на конкурс радиоуправляемую модель бронепоезда, двигавшегося по рельсам, стрелявшего из пушки и ставившего

дымовую завесу. О нем писали в газетах (“Известия”, “Комсомольская правда”, “Огонек”), а в 1935 г. приняли в авторитетное Всесоюзное общество изобретателей.

В 1941 г. в армию не взяли — глаза слабые. Башир пошел добровольцем в батальон связи, где сконструировал оригинальный шифровальный аппарат. В 1943 г. участвовал в освобождении Киева, в группе УКВ-связи.

В 1944 г. территория СССР была освобождена, и согласно постановлению о демобилизации специалистов для восстановления народного хозяйства он вернулся в Москву, на работу в ЦНИИ № 108. Как опытный радиолюбитель быстро разобрался в основных электронных схемах: триггерах, линиях задержки, регистрах, счетчиках, дешифраторах и т.д., что потом очень помогло в работе с вычислительными машинами.

В 1947 г., слушая (“подпольно”!) радио Би-би-си на самодельном приемнике, Рамеев узнал об американской электронной машине ENIAC. Заинтересовавшись, он рассказал о ней А.И. Бергу. Адмирал, как человек мудрый и истинно интеллигентный, был демократичен, доступен и очень внимателен к молодежи. Берг посоветовал обратиться к И.С. Бруку и сам ему позвонил.

Импульсивный Брук отреагировал мгновенно — в мае 1948 г. он зачислил Рамеева инженером-конструктором в Лабораторию электросистем ЭНИН АН СССР и усадил в своем кабинете — режим секретности никто не отменял.

Трудно представить, но уже через три месяца, в августе 1948 г., И.С. Брук и Б.И. Рамеев представили первый в СССР проект “Автоматическая цифровая электронная машина”, содержавший описание принципиальной схемы машины и арифметических операций в двоичной системе, управление работой машины от главного программного датчика, считывающего программу, записанную на перфоленту, и выдающего результаты на такую же ленту, а затем вводящего полученные числа с нее снова в машину для последующих вычислений.

Сам проект слишком велик, но и его оглавление дает представление об уровне проработки темы.

И.С. Брук и Б.И. Рамеев в основном реализовали принцип хранения программы в памяти. Создавалась возможность обработки команд в арифметическом устройстве машины. Все это соответствовало так называемым “принципам фон Неймана”.

Всего за год работы они подали 50 заявок на изобретения (некоторые им даже вернули — в Госкомитете по изобретениям не хватало специалистов-экспертов в этой новой отрасли). Не все 50 сделаны в 1948 г., авторы “оформляли” многие идеи предыдущих лет. Это свидетельство того, что патент № 10475 не случайное изобретение, а часть *процесса* становления ЭВМ.

С приближающимся праздником вас (и всех нас), уважаемые читатели!

Оглавление проекта “Автоматическая цифровая вычислительная машина (АЦВМ)”

I. Введение
(общие сведения о зарубежных разработках, области их применения и типах решаемых задач)
II. Общее описание АЦВМ
III. Описание отдельных элементов АЦВМ
1. Устройство для приготовления программной ленты и перевода входных данных из десятичной в двоичную систему.
2. Главный программный датчик.
3. Определитель знаков равенства и неравенства двух чисел.
4. Сумматор.
5. Умножитель.
6. Делитель.
7. Накопитель.
8. Интерполятор.
9. Устройство для перевода результатов вычисления из двоичной системы в десятичную и их печати на бумаге.
IV. Описание некоторых релейных элементов АЦВМ
1. Магнитное реле с двумя стабильными состояниями.
2. Магнитный триггер.
3. Магнитное реле, срабатывающее только при одновременном поступлении нескольких управляющих сигналов.
4. Магнитное реле, срабатывающее при поступлении одного управляющего сигнала на любой из нескольких входов.
5. Дешифратор.
V. Приложение “Таблица основных параметров быстродействующих цифровых машин, разработанных и разрабатываемых в Америке”. (Разумеется, речь идет лишь о рабочих характеристиках, приводимых в коммерческих объявлениях: скорость вычислений, величина памяти и т.п.; детали конструкции и технологии изготовления машин держались в секрете. — **Прим. авт.**)

Автоматическая цифровая вычислительная машина. Блок-схема

Конрад Цузе и его вычислительные машины

Александр Нитусов

Объемистые кропотливые расчеты статических нагрузок в конструкциях самолетов, доставшиеся 25-летнему Конраду Цузе в качестве рабочего задания, мотивировали к созданию программируемой вычислительной машины, на которую можно было бы “свалить” все это “нудное дело”. Идея оказалась “пророческой”, а сам он — одним из виднейших пионеров современной вычислительной техники.

Случай типичный — масса трудоемкой, рутинной работы, требующей выполнения в сжатые сроки, — вполне можно обобщить в виде причинно-следственной связи (вспомнив диалектику): есть причина (рабочая задача) — должно быть следствие (рабочее решение), притом эффективное. Действительно, в истории немало примеров рождения этапных изобретений или открытий в подобных ситуациях.

Знаменитый британский математик Чарльз Бэббидж, друживший в юности с сыном выдающегося астронома Гершеля, видел, на каком неподъемном количестве точных расчетов основывается наука — небесная механика. Да и хорошо знакомое британцам кораблестроение требовало все больше “математики”. Задумавшись об автоматизации вычислений, Бэббидж разработал механическую вычислительную машину, а с 1834 г. занялся ее программируемым вариантом с перфокарточным вводом информации.

В 1832 г. Семен Николаевич Корсаков — начальник статистического отдела Российского министерства внутренних дел, — устав от бесконечной работы с архивными картотеками, создал ряд перфокарточных механизмов (табуляторов) для автоматизации обработки информации и выдвинул идею “интеллектуальной машины” (искусственного интеллекта).

Полвека спустя в США Герман Холлерит, сын немецких эмигрантов, снова сделал перфокарточный табулятор (уже электромеханический — более производительный) для обработки результатов переписи населения.

Академики Сергей Лебедев и Исаак Брук, создатели первых советских электронных вычислительных машин, начинали с исследований в электроэнергетике и построения математических моделей, требовавших огромного количества непомерно сложных расчетов.

К.Цузе (1910–1995)

В 1936–38 гг. молодой берлинский инженер Конрад Цузе изготовил механическую вычислительную машину Z-1 с двоичной арифметикой, памятью и программным управлением, а в 1941 г. создал модель Z-3 — первую в мире полностью релейную цифровую вычислительную машину с программным управлением, которая успешно эксплуатировалась.

Конрад Цузе родился 22 июня 1910 г. в Берлине, а гимназию окончил неподалеку от Дрездена в г. Хойерсверда (с 1995 г. Цузе навечно зачислен в списки почетных граждан города). Затем поступил в Высшую техническую школу Берлинского университета (теперь — Берлинский технический университет), где начал изучать машиностроение, потом переключился на архитектуру, но в 1935 г. защитил диплом по специальности “инженер-строитель” и занялся расчетами статических нагрузок в авиастроительном центре Henschel Flugzeug-Werke AG (HFW) в Берлине.

Там при выполнении повторяющихся массовых статических расчетов инженер заносил данные на бланки с уже напечатанными формулами, а затем вычислял по готовой методике — почти программа для исполнителя–человека, и вполне логично было поручить ее машине.

Поработав на HFW, Цузе решил стать “свободным изобретателем”. В 1936 г. он уволился и занялся вычислительной машиной, которую начал проектировать еще будучи студентом — с 1934 г.

Молодость, работа в военной (секретной) организации и растущая культурная изоляция гитлеровской Германии не способствовали творческим контактам. К сожалению, Цузе очень долго оставался ученым-одиночкой, почти не имевшим доступа к мировым научным новинкам. Не имея понятия о структуре вычислительных устройств, он начал с нуля и самостоятельно разработал логику, дизайн машины и многое другое. Удивительные талант и упорство.

Германия всегда была одним из мировых научно-технических лидеров, и ее мощный военно-промышленный комплекс приветствовал все начинания, полезные для экономики и производства, однако для Цузе поначалу явно сделали “исключение” — серьезные специалисты сочли затею молодого инженера “дурным делом” и помогать ему никто не стал. Тем более что нацистская Германия “стояла на пороге великих свершений” (известно каких), а тут — на тебе, взрослый парень “в игрушки играет”.

В результате Цузе все делал сам. Машину он собрал в доме у родителей, зато так, как считал нужным. Помогал ему только приятель Хельмут Шраер, сын священника, студент Высшей технической школы. Работа их сблизила, и до конца жизни они поддерживали близкие отношения, хотя после войны Шраер с семьей переселился в Бразилию, где стал профессором института военной техники.

Так в 1938 г. появилась Z-1 — первая вычислительная машина, соответствовавшая в своей основе современному определению *компьютер*. Она имела двоичную арифметику, память и программное управление.

Нужно отдать должное инженерному дарованию Цузе, изучавшему теоретическую механику и сопротивление материалов, но не электротехнику. Машина состояла только из механических деталей. Их было около 30 000, в основном тонких металлических пластин, похожих на лезвия пилы. Из электрических устройств имелся только приводной электродвигатель на 1000 Вт (от пылесоса). Конструкция весила 500 кг.

Z-1 производила вычисления с плавающей точкой, при длине слова 22 бита. Данные вводились с десятичной клавиатуры и автоматически переводились в двоичную форму. Выходные данные тоже выдавались в десятичной форме. Механическая (!) оперативная память машины состояла из 64 ячеек по 22 бита каждая. Показатели длины слова были: 14 бит — мантисса, 7 бит — порядок и 1 бит — знак перед числом. Противоречивой является информация о наличии в машине перфоленты для ввода программы, сделанной из кинопленки шириной 35 мм. Сын Цузе, Хорст, в интервью, данном Компьютерной ассоциации (Computer Conservation Society) при Музее науки в Лондоне в ноябре 2010 г., утверждает (на основе анализа архива отца), что в Z-1 стоял считыватель перфоленты (см. также *www.zib.de/zuse/Inhalt/Texte/Chrono/30er/Pdf/438scan.pdf*).

Работала машина крайне ненадежно — точность полукустарного изготовления механических пластин была невысока, они часто зацеплялись и заклинивали.

Интересно, что еще в 1937 г. Шраер предложил было использовать радиолампы, но Цузе их недооценил, считая, что "железо надежнее". Кстати, Шраера "осмеяли" и в университете, когда услышали, что вычислительная машина потребует не менее 2000 ламп. Но потом он защитил диссертацию об электронном вычислителе и сам сделал экспериментальную вычислительную машину.

Z-1 вызвала некоторый интерес, но в 1939 г. Германия начала войну и Цузе призвали в армию. Он и его друзья убеждали командование в необходимости развития вычислительных машин. Шраер даже обещал разработать машину для ПВО, на что получил ответ: "Мы уже все войны выиграем, пока вы там что-то смастерите".

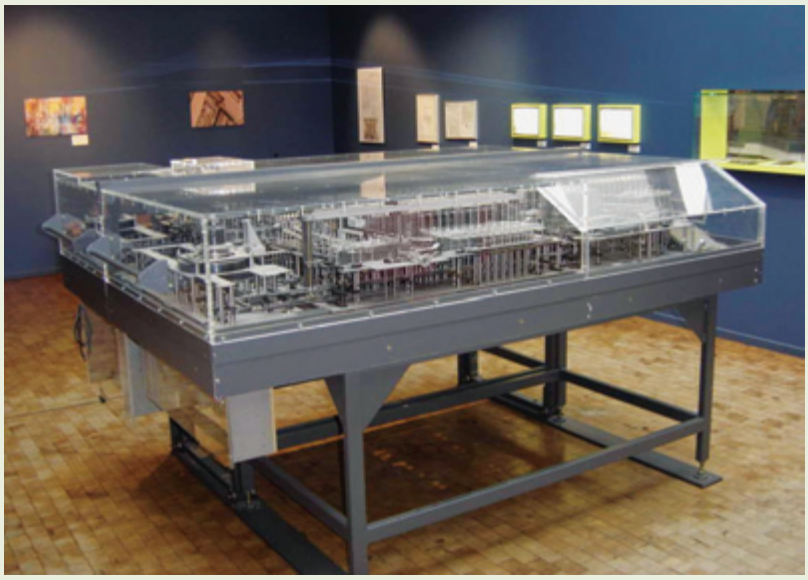
Все-таки через полгода Цузе "частично демобилизовали", обязав работать на авиацию. В 1940 г., работая на фирме HFW, он создал специализированные устройства S1 и S2, вычислявшие аэродинамические поправки к форме крыльев радиоуправляемых ракет. В S2 был встроен аналого-цифровой программно-управляемый преобразователь. По сути, S2 был первый, как бы сейчас сказали, "управляющий компьютер".

В 1940 г. уже по заказу и при помощи фирмы HFW они со Шраером выпустили следующую машину — Z-2. На этот раз с арифметическим устройством с электромеханическими телефонными реле. Модель Z-2 работала стабильнее Z-1, но тоже осталась демонстрационным экземпляром.

Однако на фирме HFW ее преимущества оценили, и Цузе основал фирму для производства программируемых компьютеров.

В 1941 г. появилась модернизированная модель Z-3. Она имела два регистра, 64 ячейки памяти и выполняла ввод/вывод, основные арифметические операции и извлечение корней числа с плавающей точкой. Центральное арифметическое устройство состояло из телефонных реле, но были внедрены и некоторые идеи Х.Шраера об электронных решениях. Возможности программирования расширились, хотя условных переходов и программных циклов в Z-3 еще не было.

Модель Z-3 считается первой в мире полностью релейной цифровой вычислительной машиной с памятью и программным управлением. В 2011 году ей исполнилось 70 лет!



Вычислительная машина Z-1 в Берлинском музее техники

Х.Шраера тоже не призвали в армию — помимо помощи Цузе, он занимался разработкой акселерометров для ракет. К 1942 г. Шраер сам сделал экспериментальную вычислительную машину со 100 электронными лампами, которая, однако, была уничтожена в конце войны. В 1944 г. он придумал электрическое устройство для перевода десятичных чисел в двоичные.

В 1942 г. Цузе приступил к созданию следующей модели Z-4. В начале 1945 г. фирма Цузе со всеми машинами (Z-1, Z-2, Z-3) была уничтожена во время англо-американских

бомбардировок, а вот частично готовую релейную Z-4 он спас, увезя ее на лошади в тихую деревню в Баварии.

Возобновить работу К.Цузе смог только в 1949 г., когда показал Z-4 профессору Эдуарду Штифелю из Швейцарского федерального технологического института в Цюрихе, и тот заказал ему один экземпляр. 12 июля 1950 г. готовую Z-4 доставили в Цюрих, где она работала очень надежно.

Почти через полвека после начала работ, в 1987–89 гг., Цузе с коллегами собрали Z-1 заново для Берлинского музея техники. Основную часть стоимости проекта (800 000 марок, или, "по-современному", 400 000 евро) предоставил концерн Siemens. В 1988 г. работа подверглась серьезной угрозе в связи с внезапным сердечным приступом у Цузе.

Занимаясь Z-4, Цузе видел, насколько сложно программирование в машинных кодах, и к 1945–46 гг. разработал (или "доработал") первый в мире язык программирования высокого уровня — Планкалкюль (Plankalkül). С конца 30-х годов он занимался также параллельными вычислениями, а с 1937 г. вел записи об искусственном интеллекте. Кроме того, Цузе в 1937 г. представил заявку на два патента на вычислительные машины, которые по сути соответствовали принципам фон Неймана[2].

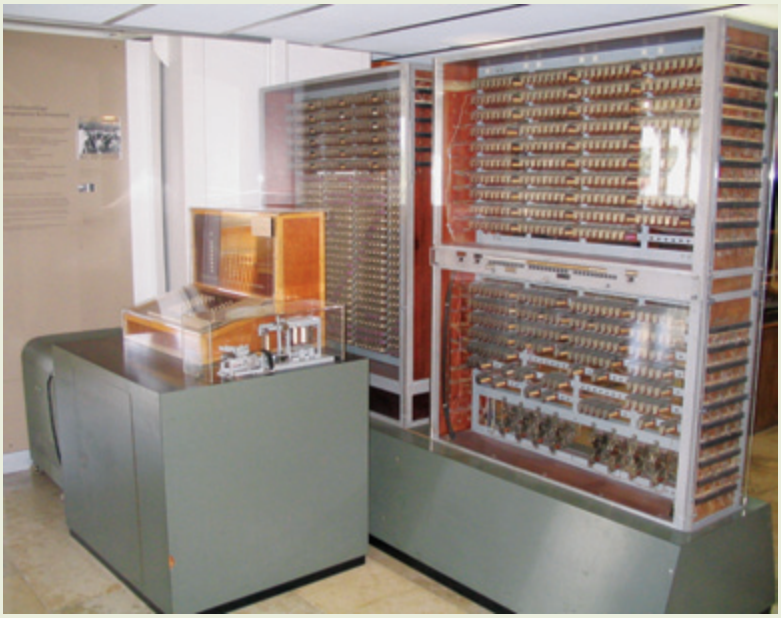
Еще в 1934 г. он предложил использовать двоичную арифметику для машинных вычислений. В СССР на эту тему первым защитил диссертацию В.Шестаков (МГУ) в 1935–37 гг., более известный на Западе Клод Шеннон (США) опубликовал аналогичные исследования в 1937–38 гг. и тогда же это сделали Накасима и Ханцзава в Японии. В Англии, США и СССР считываемые программы и устройства памяти появились в вычислительных машинах уже после Второй мировой войны.

После окончания войны Цузе остался в Германии, разрабатывая вычислительную технику для нужд ее промышленности, но спрос был неустойчив, и его фирма в конце концов стала убыточной.

В последние годы жизни Цузе вернулся к живописи. За свою жизнь он нарисовал более 500 картин, одну из которых, портрет Билла Гейтса, Цузе в марте 1995 г. подарил последнему. Портрет теперь висит у Гейтса в кабинете.

Конрад Цузе был доброжелательным, демократичным человеком. На вопрос, как он соглашался работать на гитлеровские ВВС, отвечал так: "Я не принимал нацизм, не вступал ни в какие политические организации и поначалу думал только о научной работе, но… живя в Берлине, видел, как на моих глазах сотни мирных граждан почти ежедневно погибают от бомбардировок союзников, и мне очень хотелось сделать в ответ на это какую-нибудь ракету".

История рассудит…



Вычислительная машина Z-3 в Мюнхенском политехническом музее

[2] Основные принципы построения и функционирования универсальных электронно-вычислительных машин, описанные в 1946 г. Джоном фон Нейманом, Г.Голдстайном и А.Берксом в совместной статье. — **Прим. ред.**

"ЛОМАЕМ" ГОЛОВУ

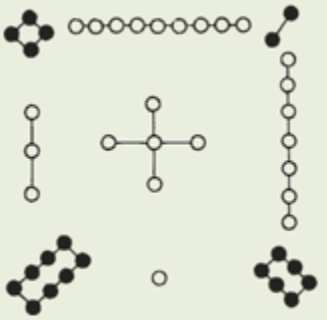
Три кучки спичек

Помня о том, что "спички детям не игрушка", предлагаем тем не менее задачу с их использованием.

Положите на стол три кучки спичек: в одну кучку — 11 спичек, во вторую — 7, в третью — 6. Перекладывая спички из одной кучки в другую, нужно сделать так, чтобы в каждой кучке было бы по 8 спичек (это возможно, так как общее число спичек — 24 — делится на 3 без остатка). При этом требуется соблюдать следующее правило: к любой кучке разрешается дополнять ровно столько спичек, сколько в ней есть. Например, если в кучке 6 спичек, то и добавлять к ней можно только 6, если в кучке 4 спички, то и добавлять к ней можно только 4. Как решить задачу?

Странное изображение

Как вы можете прокомментировать изображение, представленное на рисунке?



Примечание. Задание предназначено для учащихся начальной школы и учеников 5–7-х классов.

ДЛЯ ЭРУДИТОВ

О палиндромах

О том, что называют палиндромом, рассказано в рубрике “Семинар” в этом выпуске. Слово *палиндром* в переводе с греческого буквально означает “бегущий назад”. По-русски палиндром часто называют “перевертень” [1].

В древности палиндромам придавали магический или сакральный смысл. Самым древним из магических палиндромов считают такой: SATOR AREPO TENET OPERA ROTAS (в переводе с латыни — “Сеятель Арепо с трудом держит колеса”). Из него складывается магический квадрат, где выражение читается как вертикально, так и горизонтально, как слева направо, сверху вниз, так и наоборот:

S	A	T	O	R
A	R	E	P	O
T	E	N	E	T
O	P	E	R	A
R	O	T	A	S

Из-за магических свойств этот палиндром считали оберегом от болезней и злых духов.

Волшебный смысл палиндромов, видимо, осознавали и русские скоморохи, в своих представлениях выкрикивавшие: “На в лоб, болван”.

Широко известен и палиндром русского поэта Гавриила Романовича Державина: “Я иду съ мечемъ судия”. Многие читатели, конечно, читали сказку Алексея Николаевича Толстого “Золотой ключик, или Приключения Буратино” и помнят “диктант”, который Мальвина устроила главному герою сказки (“Она подняла к потолку хорошенькие глаза. Пишите: «А роза упала на лапу Азора». Написали? Теперь прочтите эту волшебную фразу наоборот”). Использованный палиндром сочинил другой русский поэт А.Фет. Велимир Хлебников создал целую палиндромическую поэму “Ра-зин”.

20 февраля 2002 года (сама по себе запись даты “палиндромична”!) Питер Норвиг разработал компьютерную программу, которая сгенерировала самый длинный палиндром на английском языке. Он состоит из 17 259 слов. Однако данный палиндром можно назвать таковым лишь условно, поскольку он не является логически связанным текстом. Это всего лишь набор слов, разделенных запятыми.

В заключение заметим, что самым длинным словом-палиндромом в мире считается финское *saippuakauppias* — продавец мыла, а также приведем изящный пример, принадлежащий Кириллу Решетникову: “Он дивен, палиндром, и ни морд, ни лап не видно”…

Литература

1. *Беляева Н.* Геометрия в поэзии. / Математика, № 11/2011.

“ЛОМАЕМ” ГОЛОВУ

Получить из “СОКОЛ” — “КОЛОС”

На листе бумаги нарисованы 9 клеток (*рис.* 1).

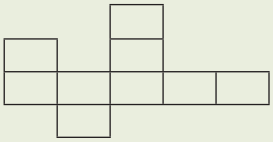


Рис. 1

На них положили 5 карточек с буквами, образующих слово СОКОЛ (*рис.* 2):

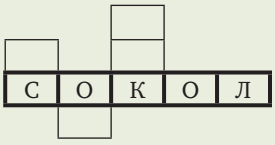


Рис. 2

Необходимо, перекладывая карточки на соседнюю свободную клетку листа, за минимальное количество перекладываний получить слово КОЛОС (*рис.* 3):

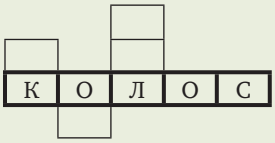


Рис. 3

Алгоритм решения задачи, пожалуйста, оформите с использованием символов “>” (перемещение карточки вправо), “<” (влево), “ V ” (вниз), “Λ” (вверх) в виде:

1. **К** Λ. 2. **О** <.

Два “географических” числовых ребуса

Решите, пожалуйста, два числовых ребуса.

Как обычно в таких головоломках, одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры.

1. **ГАВ** = **АН**$^{\text{А}}$. 2. **СССР** = **Р**$^{\text{Ф}}$.

Ответы с обоснованием присылайте в редакцию.

ВНИМАНИЕ! КОНКУРС

Конкурс № 90

В качестве заданий этого конкурса предлагаем выполнить задания для самостоятельной работы в статье “Логические и сдвиговые операции”.

Ответы с обоснованием отправьте в редакцию до 10 декабря по адресу: 121165, Москва, ул. Киевская, д. 24, “Первое сентября”, “Информатика” или по электронной почте: *vmi@1september.ru*. Пожалуйста, четко укажите в ответе свои фамилию и имя, населенный пункт, номер и адрес школы, фамилию, имя и отчество учителя информатики.

АКЦИЯ–2012

Полугодовая подписка
на электронную версию журнала

«Информатика»

200 рублей!

Каждый подписчик получает по почте именной сертификат, подтверждающий профессиональную компетентность в использовании ИКТ

Акция–2012 проводится в рамках тарифного плана «Экономичный» .
Все тарифные планы – на с. 31 этого номера

Подписка на сайте www.1september.ru