Amazon (www.amazon.com), который и сейчас остаётся самым крупным в мире.

Электронная коммерция включает в себя:

- исследование рынка;
- обмен данными и документами в электронном виде;
- денежные операции в электронной форме;
- продажу товаров, услуг и информации;
- поддержку покупателей после продажи.

Сейчас многие покупатели начинают поиски нужного товара в Интернете, а не в обычных магазинах. Чтобы привлечь внимание клиентов, фирмы размещают подробную информацию о товарах на своих сайтах, делают рассылки по электронной почте, создают сообщества в социальных сетях, организуют дискуссии на форумах. Посетителям сайтов предлагается оставить отзыв о товарах на специальной веб-странице, чтобы остальные смогли его прочитать.

Через Интернет удобно приобретать книги и электронику, авиа- и железнодорожные билеты, оплачивать услуги операторов сотовой связи.

Для пользователя всегда важно знать, что после покупки он не окажется один на один со своими проблемами и сможет получить консультацию. Поэтому поддержка покупателей — тоже важный элемент электронной коммерции. На сайтах производителей оборудования (например, жёстких дисков, принтеров, сканеров и т. п.) всегда можно найти всю документацию по настройке и обслуживанию устройств и бесплатно скачать самые новые драйверы. На форумах фирм-изготовителей пользователи получают консультацию службы технической поддержки и делятся друг с другом решениями возникших проблем.

Бизнес в Интернете предоставляет дополнительные **возможности для компаний**:

- расширение сферы влияния: фирмы любого размера могут принимать заказы со всего мира;
- увеличение конкурентоспособности: быстрая реакция на претензии клиентов и изменение ситуации на рынке;
- индивидуальный подход (система собирает информацию о клиенте и пытается предложить именно те товары, которые он чаще покупает);
- уменьшение затрат: не нужно арендовать помещение для магазина и нанимать много сотрудников.

Покупатели тоже получают серьезные преимущества:

- выбор товаров и услуг из большого количества вариантов;
- легко сравнить разные предложения;
- можно узнать отзывы других пользователей;
- можно заказывать и оплачивать товары в удобное время;
- цены на товары и услуги обычно ниже, чем в обычных магазинах.

Интернет-магазины

Всё больше людей используют **интернет-магазины** — веб-сайты, которые рекламируют товары или услуги, принимают заказы на покупку, предлагают варианты оплаты и получения заказа. Выбрав товары в интернет-магазине, пользователь может «положить их в корзину», т. е. включить в список для приобретения. Когда состав покупки полностью определен, оформляется заказ: покупатель указывает свои данные, способ оплаты и доставки. На многих сайтах возможен заказ товара по телефону.

Оплата выполняется разными способами:

- через банковскую карту, пригодную для оплаты в Интернете (Visa, MasterCard);
- банковским переводом (например, через Сбербанк);
- почтовым переводом;
- с помощью электронных платёжных систем (электронными деньгами);
- наличными при получении товара;
- через отправку платного SMS-сообщения (для небольших покупок).

Для «вещественных» товаров существуют три способа **доставки**:

- курьерская доставка по указанному адресу;
- почтовая доставка;
- «самовывоз» с пунктов выдачи товара (в некоторых городах).

Товары в электронной форме (программы, коды доступа, тексты, статьи, фотографии и т. п.) обычно высылаются по электронной почте или покупателю предоставляется персональная ссылка на нужный файл на сервере фирмы-продавца.

Для управления интернет-магазином используют специальное программное обеспечение, например кроссплатформенные системы *1С-Битрикс* (www.1c-bitrix.ru) и *osCommerce* (www.oscommerce.com).

Ещё один вид электронной коммерции — **интернет-аукционы** («онлайновые аукционы»), в которых фирма-организатор — это

просто посредник между продавцом и покупателем. Любой желающий может делать ставки, используя веб-сайт аукциона. Когда интернет-аукцион завершен, покупатель, сделавший наибольшую ставку, должен перевести деньги продавцу, а продавец обязан выслать товар покупателю по почте. Крупнейший интернет-аукцион — *eBay* (www.ebay.com).

При покупках в Интернете нужно соблюдать некоторую осторожность. В первую очередь, обращайте внимание на то, чтобы на сайте магазина были указаны регистрационные данные и юридический адрес организации, контактные телефоны, условия доставки и возврата покупок. Сайт, сделанный непрофессионально, — серьёзный повод для отказа от покупки. Полезно заранее поискать в Интернете отзывы покупателей о выбранном магазине. Надежнее всего обращаться в известные Интернет-магазины (например, ozon.ru), которые дорожат своей репутацией.

Электронные платежные системы

Электронная торговля не была бы столь удобной, если бы не было возможности оплачивать покупку прямо в Интернете, не выходя из дома. Для этого должны были появиться **электронные деньги**, которые можно свободно купить и продать. Системы расчётов электронными деньгами называются **электронными платёжными системами**. Среди российских платёжных систем наиболее известны *WebMoney* (www.webmoney.ru) и *Яндекс.Деньги* (money.yandex.ru). Популярная международная почтовая система — *PayPal* (www.paypal.com).

В платёжной системе можно завести **электронный кошелёк**, который пополняется с помощью специальных карт оплаты или через терминалы. Из такого кошелька можно оплачивать коммунальные услуги, сотовую связь и Интернет, покупать авиа- и железнодорожные билеты, товары в интернет-магазинах. По условиям пользовательского соглашения кошелёк системы *Яндекс. Деньги* нельзя использовать для коммерческой деятельности (например, для получения оплаты за выполненную работу), иначе его может заблокировать служба безопасности[1].

Пользователи могут переводить электронные деньги не только на счета интернет-магазинов, но и на кошельки других пользователей. Чтобы убедиться в том, что вы не ошиблись при вводе номера кошелька и переводите деньги именно тому, кому нужно, применяют перевод с кодом протекции. **Код протекции** — это некоторое число, которое должен ввести получатель для получения перевода на свой счет. Этот код можно сообщить по электронной почте или по телефону. Если получатель вводит неверный код несколько раз подряд или истекает срок действия перевода, средства возвращаются в кошелёк отправителя.

[1] Для того чтобы принимать оплату через Яндекс.Деньги, Интернет-магазины получают специальное разрешение от администрации сервиса.

При необходимости можно вывести средства из электронного кошелька, т. е. получить их в виде наличных денег в банке (за вычетом небольшой комиссии).

С электронными кошельками можно работать в браузере (через веб-интерфейс сайта) или с помощью специальной программы, которая устанавливается на компьютер пользователя.

Вопросы и задания

1. Что такое электронная коммерция?
2. Какие направления включает электронная коммерция?
3. Какие преимущества предоставляет электронная торговля для продавцов и покупателей?
4. Что такое интернет-аукцион?
5. Зачем используется код протекции при переводе электронных денег?
6. Что такое интернет-магазин? Объясните на примере.
7. Как можно оплатить покупку в интернет-магазине? Как доставляются покупки?
8. Что такое электронные деньги? Чем они, на ваш взгляд, отличаются от «обычных» денег?

Подготовьте сообщение

а) «Интернет-магазины: за и против»
б) «Интернет-аукционы»
в) «Электронные платёжные системы»

§ 53
Право и этика в Интернете

Интернет и закон

Как только в сети появилась купля-продажа товаров и услуг, возникла необходимость регулировать эти отношения с помощью закона, защищать интересы пользователей и предотвращать мошенничество.

Интернет — это не организация, он не принадлежит ни одной стране, развивается во многом стихийно и не может быть юридическим лицом. В связи с этим возникает множество проблем, с которыми юристы раньше не сталкивались. Например, не вполне ясно:

- несёт ли провайдер ответственность за действия пользователей, которым он предоставляет доступ в Интернет (в том числе за нарушения авторских прав);
- можно ли признавать доказательствами цифровые документы (сообщения электронной почты, рисунки, звукозаписи, видео);
- как доказать условия заключённой сделки, если фирма может в любой момент изменить условия договора на сайте;
- какую ответственность несут платёжные системы перед государством и пользователями.

Соблюдение **авторских прав**, т. е. прав автора на результаты своего интеллектуального труда, — один из самых актуальных правовых вопросов Интернета. Практически вся информация на сайтах защищается авторским правом: программное обеспечение, тексты, рисунки и фотографии, музыка и видеофильмы. Часто на веб-страницах публикуются условия использования материала (англ. *terms of use*), где указывается, можно ли сохранять и копировать материал, вставлять его в другие документы, распечатывать. Если такой информации нет, следует получить разрешение на использование материала, отправив сообщение веб-мастеру (автору сайта) по электронной почте.

На своём веб-сайте **можно без разрешения** размещать:

- гиперссылки на другие сайты;
- бесплатную графику;
- произведения авторов, со дня смерти которых прошло более 70 лет;
- официальные документы.

Без разрешения нельзя:

- копировать содержание других сайтов;
- объединять информацию из разных источников для создания «собственного» документа;
- изменять чужой текст или изображение;
- размещать любые изображения с других сайтов, о которых явно не написано, что они бесплатные.

В Гражданском кодексе Российской Федерации (ГК РФ) определяется, что можно без согласия автора использовать его произведения в научных, учебных или культурных целях (статья

1274). При этом обязательно указать имя автора и источник (книгу, статью, сайт). Например, разрешается:

- цитировать произведения (для того, чтобы подтвердить или опровергнуть какую-то мысль автора) в объёме, оправданном целью цитирования;
- использовать произведения и их отрывки в учебных материалах в объёме, оправданном поставленной целью;
- использовать произведения для создания пародии или карикатуры.

Использование чужого произведения (например, текста, музыки или видеозаписи) как составной части в своих работах является нарушением авторского права независимо от объёмов (например, нельзя включить в свой фильм даже 1 секунду из другого фильма или звукозаписи). При этом не имеет значения, что вы не получили от этого коммерческой выгоды (это влияет только на размер штрафа).

Переработка чужого материала (например, наложение текста, графики, звука, монтаж видео) — это серьёзное нарушение права автора на неприкосновенность произведения (статьи 1255 и 1266 ГК РФ), за это в законе предусмотрены значительные штрафы.

Незаконный доступ к информации — это уголовное преступление, за которое в Уголовном кодексе Российской Федерации предусмотрен штраф или лишение свободы на срок до двух лет (статья 272). Если ваш сайт (или учётную запись на сайте) взломали или вы стали жертвой интернет-мошенничества, нужно обращаться в **отдел по борьбе с компьютерными преступлениями** (отдел «К») полиции.

Нетикет

На ранних этапах развития, когда к Интернету были подключены только научные центры и университеты, общение основывалось на взаимном уважении и доверии пользователей. В результате сложились неофициальные правила общения, которые называются **сетевым этикетом** или **нетикетом** (фр. *netiquette*).

Несмотря на то что при общении в Интернете вы, как правило, не видите собеседника, нужно вести себя так, как будто вы говорите с человеком лично: не пишите то, что вы не смогли бы сказать ему в лицо; не оскорбляйте собеседника, не ругайтесь. Перед тем как послать сообщение, прочтите его внимательно ещё раз и поставьте себя на место получателя.

Передавая информацию по открытым каналам, помните, что копии всех ваших сообщений могут храниться, например, у провайдера. Не посылайте информацию, доступ к которой ограничен. Уважайте авторские права, не используйте чужой материал без разрешения. Уважайте тайну переписки: если вы хотите опубликовать личное сообщение, нужно спросить разрешения у автора.

В сообщениях **электронной почты** и **форумов**:

- пишите кратко и точно, цените время других людей;
- не пишите слова заглавными буквами (это воспринимается как крик или визг);
- не используйте сленг, пишите грамотно, не пропускайте пробелы и знаки препинания;
- для передачи тона письма используйте **смайлики** (англ. *smiley*) — значки для обозначения эмоций, например:

  :-) или :) — улыбка;
  :-(или :(— несчастное лицо, сожаление или разочарование;
  ;-) или ;) — подмигивающее лицо, слова не следует понимать слишком серьезно;

- цитируйте те высказывания, на которые отвечаете (собеседник может забыть содержание предыдущего письма);
- не распространяйте спам — нежелательную рекламу.

Посылая сообщения **по электронной почте**:

- обязательно пишите тему сообщения, отражающую содержание письма;
- ставьте подпись (имя и фамилию) в конце письма;
- не посылайте большие файлы без согласия получателя.

Участвуя **в форумах**:

- сначала прочитайте список часто задаваемых вопросов и прошлые сообщения, возможно, вы найдете там ответ на свой вопрос;
- отправляя сообщение, осознайте, что многие увидят ваш текст;
- не отклоняйтесь от темы обсуждения;
- не участвуйте во **флейме** (от англ. *flame* — огонь, пламя) — так называется «спор ради спора», словесная война; обычно за флейм наказывают модераторы форума;
- не разжигайте **холивары** (от англ. *holy war* — «священная война») — так называется спор о двух идеях, каждая из которых имеет своих сторонников (например, что лучше: Windows или Linux, Паскаль или Си и т. п.).

Общаясь **в чатах**:

- не перебивайте собеседника;
- не обижайтесь, если незнакомые люди не хотят с вами разговаривать;
- не пытайтесь выведывать личную информацию;
- уважайте анонимность, не разглашайте реальное имя другого участника без разрешения, если вы его знаете;
- будьте снисходительны к ошибкам других;
- не обижайтесь, если собеседник неожиданно покинул чат.

Вопросы и задания

1. Какие юридические проблемы возникают в связи с куплей-продажей услуг в Интернете?
2. В каком случае можно размещать на своем веб-сайте чужой материал?
3. В каком случае можно бесплатно использовать произведения без согласия автора? Какие ограничения нужно при этом соблюдать?
4. Расскажите о наиболее распространённых нарушениях авторских прав в Интернете.
5. Является ли взлом персональной странички в социальной сети преступлением? Почему?
6. Что можно сделать, если вас обманули мошенники в Интернете?
7. Что такое нетикет?
8. Можно ли опубликовать на форуме личное сообщение?
9. Какие правила рекомендуется соблюдать в сообщениях электронной почты?
10. Как нужно вести себя в форумах и чатах?
11. Как в электронном письме можно передать свое настроение?
12. Почему при пересылке больших файлов нужно спросить согласие получателя?
13. Объясните значение слов «флейм» и «холивар».

Подготовьте сообщение

а) «Интернет и право»
б) «Авторские права в Интернете»
в) «Сетевой этикет»

Практические работы к главе 7

Работа № 23 «Тестирование сети»
Работа № 24 «Сравнение поисковых систем»

ЭОР к главе 7 на сайте ФЦИОР (http//fcior.edu.ru)

- Глобальные компьютерные сети
- Архитектура Интернета
- Адресация в Интернете
- Технология трансляции сетевых адресов
- Технология WWW
- Протоколы передачи данных в сети Интернет
- Контроль знаний: история Всемирной паутины
- Службы Интернета
- Поиск информации
- Поиск информации в Интернете
- Законодательство РФ «Об информации, информационных технологиях и о защите информации»

Самое важное в главе 7

- Компьютерная сеть — это группа компьютеров, соединённых линиями связи. Сети предназначены для ускорения обмена данными и совместного использования ресурсов. Вместе с тем объединение компьютеров в сеть снижает безопасность данных.
- Сервер — это компьютер, предоставляющий свои ресурсы в общее использование. Клиент — это компьютер, использующий ресурсы сервера. Понятия «сервер» и «клиент» применяются также к программам.
- Протокол — это набор правил и соглашений, определяющих способ и порядок обмена данными в сети. Компьютеры могут обмениваться данными только тогда, когда они поддерживают общий протокол.
- В современных сетях используется пакетная передача данных, при которых нагрузка на сеть становится более равномерной.
- Существуют три основные структуры (топологии) сетей: общая шина, звезда и кольцо. В современных локальных сетях, как правило, используется схема «звезда».
- Интернет — это глобальная компьютерная сеть, которая использует семейство протоколов TCP/IP.
- Любой компьютер, подключенный к сети Интернет, имеет собственный IP-адрес. Если у компьютера есть несколько интерфейсов (сетевых карт или адаптеров беспроводной связи), каждый из них может иметь свой IP-адрес.
- Система доменных имён (DNS) позволяет сопоставить IP-адресу символьное имя.
- Адрес ресурса (URL) содержит протокол, имя сервера, каталог и имя документа (файла).

Глава 8
Алгоритмизация и программирование

§ 54
Алгоритм и его свойства

Мухаммед ал-Хорезми (ок. 783–ок. 850 гг.)

Что такое алгоритм?

Происхождение слова «алгоритм» связывают с именем учёного Мухаммеда ал-Хорезми (перс. خوارزمی [al-Khwārazmī]), который описал десятичную систему счисления (придуманную в Индии) и предложил правила выполнения арифметических действий с десятичными числами.

Алгоритм — это точное описание порядка действий, которые должен выполнить исполнитель для решения задачи.

Здесь **исполнитель** — это устройство или одушёвленное существо (человек), способное понять и выполнить команды, составляющие алгоритм.

Человек как исполнитель часто действует неформально, по-своему понимая команды. Несмотря на это, ему тоже часто приходится действовать по тому или иному алгоритму. Например, рецепт приготовления какого-либо блюда можно считать алгоритмом. На уроках русского языка, выполняя разбор слова или предложения, вы тоже действуете по определённому алгоритму. Много различных алгоритмов в математике (постарайтесь вспомнить известные вам). На производстве рабочий, вытачивая деталь в соответствии с чертежом, действует по алгоритму, который разработал технолог. И таких примеров может быть множество.

В информатике рассматривают только **формальных исполнителей**, которые не понимают (и не могут понять) смысл команд. К этому типу относятся все технические устройства, в том числе и компьютер.

Каждый формальный исполнитель обладает собственной **системой команд**. В алгоритмах для такого исполнителя нельзя использовать команды, которых нет в его системе команд.

Свойства алгоритма

- **Дискретность** — алгоритм состоит из отдельных команд (шагов), каждая из которых выполняется за конечное время.
- **Детерминированность (определённость)** — при каждом запуске алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат.
- **Понятность** — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен.
- **Конечность (результативность)** — для корректного набора данных алгоритм должен завершаться через конечное время с вполне определённым результатом (результатом может быть и сообщение о том, что задача не имеет решений).
- **Корректность** — для допустимых исходных данных алгоритм должен приводить к правильному результату.
- **Массовость** — алгоритм, как правило, предназначен для решения множества однотипных задач с различными исходными данными.

Эти свойства не равноправны. Дискретность, детерминированность и понятность — фундаментальные свойства алгоритма, т. е. ими обладают все алгоритмы для формальных исполнителей. Остальные свойства можно рассматривать как требования к «правильному» алгоритму.

Иными словами, алгоритм получает на вход некоторый дискретный входной объект (например, набор чисел или слово) и обрабатывает входной объект по шагам (дискретно), строя промежуточные дискретные объекты. Этот процесс может закончиться или не закончиться. Если процесс выполнения алгоритма заканчивается, то объект, полученный на последнем шаге работы, является результатом работы алгоритма при данном входе. Если процесс выполнения не заканчивается, говорят, что алгоритм зациклился. В этом случае результат его работы не определён.

Способы записи алгоритмов

Алгоритмы можно записывать разными способами:

- **на естественном языке**, обычно такой способ применяют, записывая основные идеи алгоритма на начальном этапе;
- на **псевдокоде**, так называется смешанная запись, в которой используется естественный язык и операторы какого-либо языка программирования; в сравнении с предыдущим вариантом такая запись гораздо более строгая;
- в виде **блок-схемы** (графическая запись);
- в виде **программы** на каком-либо языке программирования.

Мы будем записывать алгоритмы в виде программы на двух языках программирования: на **языке Паскаль** (версия FreePascal) и на алгоритмическом языке **системы КуМир**, который называют **школьным алгоритмическим языком**, а также в некоторых случаях — на **псевдокоде**.

Вопросы и задания

1. Что такое алгоритм?
2. Что такое исполнитель?
3. Чем отличаются формальные и неформальные исполнители?
4. Что такое система команд исполнителя? Придумайте исполнителя с некоторой системой команд.
5. Перечислите и объясните свойства алгоритма.
6. Какие существуют способы записи алгоритмов? Какие из них, по вашему мнению, чаще применяются на практике? Почему?

Подготовьте сообщение

а) «История слова алгоритм»
б) «Свойства алгоритма»
в) «Способы записи алгоритмов»

§ 55 Простейшие программы

Пустая программа

Пустая программа — это программа, которая ничего не делает, но удовлетворяет требованиям выбранного языка программирования. Она полезна, прежде всего, для того, чтобы понять общую структуру программы. Далее мы будем слева приводить программу на школьном алгоритмическом языке, а справа — эквивалентную программу на языке Паскаль.

```
алг Куку
нач
  | основная программа
кон
```

```
program qq;
begin
  { основная программа }
end.
```

Программа на школьном алгоритмическом языке начинается **ключевым словом алг** (сокращение от слова «алгоритм»), за которым записывают название алгоритма. Название может содержать русские и латинские буквы, цифры, знак подчёркивания «_» и даже пробелы, но не может начинаться с цифры. Между ключевыми словами **нач** и **кон** размещают **операторы** (команды) программы (**тело программы**). После вертикальной черты записывают **комментарии** — пояснения, которые не обрабатываются транслятором.

В языке Паскаль в имени программы нельзя использовать русские буквы и пробелы. Все ключевые слова записываются на английском языке. После имени программы ставится точка с запятой. Ключевые слова **begin** и **end** ограничивают тело программы, после слова **end** ставится точка. Комментарии заключают в фигурные скобки.

Вывод текста на экран

Напишем программу, которая выводит на экран такие строки:

```
2+2=?
Ответ: 4
```

Вот как она выглядит:

```
алг qq
нач
  вывод '2+'
  вывод '2=?', нс
  вывод 'Ответ: 4'
кон
```

```
program qq;
begin
  write('2+');
  writeln('2=?');
  write('Ответ: 4')
end.
```

Команда `вывод` в школьном алгоритмическом языке **выводит** на экран символы, заключённые в апострофы или в кавычки. Для перехода на новую строку в **списке вывода** нужно указать `нс` (новая строка).

В языке Паскаль для вывода данных используют процедуру `write` (без перехода на новую строку) или `writeln` (после окончания вывода происходит переход на новую строку). Параметры этих процедур заключаются в круглые скобки, а символьные строки — в апострофы. Каждый оператор заканчивается точкой с запятой, перед словом **end** можно её не ставить.

Переменные

Напишем программу, которая выполняет сложение двух чисел:

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

Программу на псевдокоде (смеси русского и школьного алгоритмического языков) можно записать так:

```
алг Сумма
нач
  ввести два числа
  сложить их
  вывести результат
кон
```

Компьютер не может выполнить псевдокод, потому что команд «ввести два числа» и ей подобных нет в его системе команд. Поэтому нам нужно «расшифровать» все такие команды через операторы языка программирования.

В отличие от предыдущих задач, здесь требуется хранить данные в памяти. Для этого используют переменные.

Переменная — это величина, которая имеет имя, тип и значение. Значение переменной может изменяться во время выполнения программы.

Переменная определяет область памяти, где хранится только одно значение. При записи в неё нового значения «старое» стирается, и его уже никак не восстановить.

Переменные в программе необходимо объявлять[1]. При **объявлении** указывается **тип** переменной и её **имя (идентификатор**, от слова «идентифицировать» — отличать один объект от другого). **Значение** переменной сразу после объявления неопределённое: переменной выделяется некоторая область памяти, и там могло быть до этого записано любое число.

Вот так объявляются целочисленные переменные (в которых могут храниться только целые значения) с именами *a*, *b* и *c*:

```
цел a, b, c
```

```
var a, b, c: integer;
```

[1] В некоторых языках программирования, например в языке Python, переменные не объявляются. Но это может привести к ошибкам, которые трудно обнаружить.

В языке Паскаль описание переменных начинается с ключевого слова **var**, после него записывают список переменных и в конце через двоеточие — их тип, в данном случае **целочисленный** (англ. **integer**).

Имена переменных строятся по тем же правилам, что и имена программ. В языке Паскаль можно использовать в именах латинские буквы (строчные и заглавные буквы не различаются), цифры (но имя не может начинаться с цифры, иначе транслятору будет сложно различить, где начинается имя, а где — число) и знак подчёркивания «_».

В школьном алгоритмическом языке в именах разрешены кроме перечисленных символов ещё пробелы и русские буквы, причём строчные и заглавные буквы различаются (поэтому `x` и `X` — это разные имена переменных).

Желательно давать переменным «говорящие» имена, чтобы можно было сразу понять, какую роль выполняет та или иная переменная.

Тип переменной нужен для того, чтобы:

- определить область допустимых значений переменной;
- определить допустимые операции с переменной;
- определить, какой объём памяти нужно выделить переменной и в каком формате будут храниться данные (вспомните, что целые и вещественные числа хранятся в компьютере по-разному, см. главу 4);
- предотвратить случайные ошибки; например, при попытке записать символ в целую переменную выдаётся сообщение об ошибке[1].

В алгоритмическом языке можно при объявлении задать начальные значения переменных:

```
цел a, b=1, c=55
```

Значение переменной `a` осталось неопределённым.

Большинство трансляторов Паскаля автоматически заполняют нулями все переменные основной программы, однако лучше не надеяться на это и явно задавать начальные значения всех переменных.

[1] Некоторые языки, например язык Си, позволяют записывать в переменную значение другого типа, но вся ответственность за результат ложится на программиста.

Приведём полную программу сложения двух чисел:

```
алг Сумма
нач
  цел a, b, c
  ввод a, b
  c:=a+b
  вывод c
кон
```

```
program Sum;
var a, b, c: integer;
begin
  read(a, b);
  c:=a+b;
  write(c)
end.
```

Эта программа, как и любая программа на любом алгоритмическом языке, «складывается» из двух составляющих: данных (переменных) и алгоритмов работы с ними, которые записываются с помощью операторов — команд языка программирования.

Оператор `ввод` (в Паскале — процедура `read`) предназначен для **ввода** данных (например, с клавиатуры).

Оператор, содержащий символы «:=», — это **оператор присваивания**, с его помощью изменяют значение переменной. Он выполняется следующим образом: вычисляется выражение справа от символов «:=», а затем результат записывается в переменную, записанную слева от символов «:=». Поэтому, например, оператор

```
i:=i+1                    i:=i+1;
```

увеличивает значение переменной i на 1.

У приведённой выше программы сложения чисел есть два недостатка:

1) перед вводом данных пользователь не знает, что от него требуется (сколько чисел нужно вводить и каких);
2) результат выдаётся в виде числа, которое означает неизвестно что.

Хотелось бы, чтобы диалог программы с пользователем выглядел так:

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

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

```
вывод a, '+', b, '=', c          write(a, '+', b, '=', c);
```

Обратите внимание, что имена переменных записаны без апострофов.

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

```
вывод a, '+', b, '=', a+b        write(a, '+', b, '=', a+b);
```

В обоих языках (в среде КуМир — начиная с версии 2.0) можно использовать **форматный вывод**: после двоеточия указать общее количество знакомест, отводимое на число. Например, программа

```
a:=123                           a:=123;
вывод a:5                        write( a:5 );
```

выведет значение целой переменной a, заняв ровно 5 знакомест:

␣␣123

Поскольку само число занимает только 3 знакоместа, перед ним выводятся два пробела, которые здесь мы обозначили как «␣».

Вопросы и задания

1. Сравните структуру программ на языке Паскаль и школьном алгоритмическом языке.
2. Что такое идентификатор?
3. Чем различаются правила построения имён в школьном алгоритмическом языке и в Паскале?
4. Как записываются комментарии на этих языках? Подумайте, как комментирование можно использовать при поиске ошибок в алгоритме.
5. Сравните операторы вывода в КуМире и в Паскале. Как выполняется переход на новую строку?
6. Что такое переменная? Как вы думаете, зачем нужно объявлять переменные?
7. Зачем нужен тип переменной? Почему нельзя записывать в переменную значение другого типа?
8. Какое значение записано в переменной сразу после объявления? Можно ли его использовать?
9. Как задать начальные значения переменных? Сравните школьный алгоритмический язык и Паскаль.
10. Что такое оператор присваивания?
11. Почему следует выводить на экран подсказку перед вводом данных?

12. Подумайте, когда можно вычислять результат прямо в операторе вывода, а когда нужно заводить отдельную переменную.
13. Что такое форматный вывод? Как вы думаете, где он может быть полезен?

Подготовьте сообщение

а) «Операторы вывода в языке Си»
б) «Операторы вывода в языке Python»

Задачи

1. Используя оператор вывода, постройте на экране следующие рисунки из символов:

```
  Ж      Ж       Ж       Ж        Ж  Ж   Ж
 ЖЖЖ    ЖЖ       Ж       Ж       ЖЖ  ЖЖ ЖЖ
ЖЖЖЖЖ  ЖЖЖЖЖЖ  ЖЖЖЖЖ     ЖЖЖ     ЖЖЖЖЖ
 Ж Ж       ЖЖ   Ж Ж Ж   ЖЖЖЖЖ    ЖЖ  ЖЖ
ЖЖЖ         Ж  ЖЖЖЖЖ  ЖЖЖЖЖЖЖ   Ж    Ж
```

2. Выберите правильные имена переменных (для школьного алгоритмического языка и Паскаля):

1	Vasya	СУ-27	@mail_ru
ml1	Петя	СУ_27	lenta.ru
lm	Митин брат	_27	"Pes barbos"
m 1	Quo vadis	СУ(27)	<Ладья>

3. Пусть a и b — целые переменные. Что будет выведено в результате работы фрагмента программы?

а)
```
цел a=5, b=3                     a:=5; b:=3;
вывод a, '=Z(', b, ')'           write(a,'=Z(',b,')');
```
б)
```
цел a=5, b=3                     a:=5; b:=3;
вывод 'Z(a)=', '(b)'             write('Z(a)=','(b)');
```
в)
```
цел a=5, b=3                     a:=5; b:=3;
вывод 'Z(',a,')=(',a+b,')'       write('Z(',a,')=(',a+b,')');
```

4. Запишите оператор для вывода значений целых переменных $a = 5$ и $b = 3$ в следующем формате:
   а) 3+5=?
   б) Z(5)=F(3)
   в) a=5; b=3;
   г) Ответ: (5;3)

5. Попробуйте определить, что делает следующий фрагмент программы, где x и y — целые переменные:

```
x:=x-y                           x:=x-y;
y:=x+y                           y:=x+y;
x:=y-x                           x:=y-x;
```

§ 56 Вычисления

В курсе программирования можно выделить две важнейшие составляющие — алгоритмы и способы организации данных[1]. В этом параграфе мы познакомимся с простейшими типами данных в школьном алгоритмическом языке и в Паскале. В следующих разделах рассматриваются основные алгоритмические конструкции: ветвления, циклы, подпрограммы. В конце главы подробно изучаются сложные (составные) типы данных: массивы, символьные строки, работа с файлами.

Типы данных

Любая переменная имеет какой-либо тип, т. е. может хранить данные только того типа, который был указан при её объявлении.

В школьном алгоритмическом языке используются такие типы:

- **цел** — целые значения;
- **вещ** — вещественные значения;
- **лог** — логические значения (**да** или **нет**);
- **сим** — символ;
- **лит** — литерная переменная (символьная строка, т. е. цепочка символов).

В языке Паскаль типов немного больше. Например, кроме типа **integer** есть несколько других типов переменных для хранения целых чисел:

- **byte** — целые числа в диапазоне[2] 0..255;
- **shortint** — целые числа в диапазоне –128..127;
- **word** — целые числа в диапазоне 0..65535;
- **longint** — целые числа в диапазоне –2147483648..2147483647.

На переменную типа **byte** и **shortint** в памяти выделяется 1 байт, на переменную типа **word** — 2 байта, на переменную типа **longint** — 4 байта. Размер переменных типа **integer** зависит от версии языка (2 или 4 байта).

[1] Знаменитая книга швейцарского специалиста Никлауса Вирта, разработчика языков Паскаль, Модула-2 и Оберон, так и называлась «Алгоритмы + структуры данных = программы».

[2] Диапазон состоит из всех целых чисел от минимального до максимального значения (включая обе эти границы).

Для хранения вещественных переменных тоже существует несколько типов:

- **single** — число одинарной точности (4 байта);
- **real** — классический тип языка Паскаль (6 байтов);
- **double** — число двойной точности (8 байтов);
- **extended** — число расширенной точности (10 байтов).

Как мы обсуждали в главе 4, большинство вещественных чисел хранится в памяти неточно, и в результате операций с ними накапливается вычислительная ошибка. Поэтому для работы с целочисленными данными не нужно использовать вещественные переменные.

Логические переменные в Паскале относятся к типу **boolean** и принимают значение `True` (истина) или `False` (ложь). Несмотря на то, что теоретически для хранения логического значения достаточно одного бита памяти, обычно такая переменная занимает в памяти один байт (или даже несколько байтов). Так как процессор может читать и записывать в память только целые байты, операции с логическими переменными в этом случае выполняются быстрее.

Переменные типа **char** могут хранить один символ (точнее, его код) размером 1 байт. Символьные строки относятся к типу **string** (во многих версиях Паскаля длина строки не может быть больше 255 символов).

Все перечисленные типы данных, кроме символьных строк, относятся к простым типам — они представляют собой единые объекты, в которых нельзя выделить составные части. Символьные строки — пример сложного типа данных, они состоят из отдельных символов. Далее мы познакомимся с другими сложными типами данных: массивами, файлами, структурами.

Арифметические выражения и операции

В языках программирования используется линейная запись арифметических выражений (без многоэтажных дробей). В школьном алгоритмическом языке выражение записывается в одну строку, а в Паскале запись можно переносить на следующие строки.

Арифметические выражения могут содержать константы (постоянные значения), имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например:

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

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

- действия в скобках;
- умножение и деление, слева направо;
- сложение и вычитание, слева направо.

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

Если в выражение входят переменные разных типов, в некоторых случаях происходит автоматическое приведение типа к более «широкому». Например, результат умножения целого числа на вещественное — это вещественное число. Переход к более «узкому» типу автоматически не выполняется, поэтому, например, вещественное значение нельзя записать в целую переменную. Нужно помнить, что результат деления (операции «/») — это вещественное число, даже если делимое и делитель — целые и делятся друг на друга нацело[1].

Часто нужно получить целый результат деления целых чисел и остаток от деления. В этом случае в школьном алгоритмическом языке используют функции div и mod, а в Паскале — одноимённые операции (они имеют такой же приоритет, как умножение и деление):

```
d:= 85                          d:= 85;
a:=div(d,10) | = 8              a:=d div 10; { = 8 }
b:=mod(d,10) | = 5              b:=d mod 10; { = 5 }
```

Нужно учитывать, что для отрицательных чисел эти операции по-разному выполняются в разных языках. Например, внешне следующие программы аналогичны:

```
вывод div(-7,2), ','          write(-7 div 2, ',');
вывод mod(-7,2)               write(-7 mod 2);
```

Но программа на школьном алгоритмическом языке выведет результат «–4,1», а программа на Паскале — «–3,–1». Дело в том, что с точки зрения теории чисел остаток — это неотрицательное число, поэтому $-7 = (-4) \cdot 2 + 1$, т. е. частное от деления (–7) на 2 равно –4, а остаток равен 1. Поэтому в среде КуМир эти операции выполняются математически правильно. В то же время

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

во многих языках (например, в Паскале и в Си) при целочисленном делении используется модуль числа, а затем к частному и остатку добавляется знак «минус»:

$$7 = 3 \cdot 2 + 1 \Rightarrow -7 = (-3) \cdot 2 - 1.$$

При таком подходе частное от деления (–7) на 2 равно –3, а результат операции mod равен –1.

В школьном алгоритмическом языке есть операция возведения в степень, которая обозначается двумя звездочками: «**». Например, выражение $y = 2x^2 + z^3$ запишется так:

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

Возведение в степень имеет более высокий приоритет, чем умножение и деление. В языке Паскаль операции возведения в степень нет.

Вещественные значения

При записи **вещественных чисел** в программе целую и дробную части разделяют не запятой (как принято в отечественной математической литературе), а точкой. Например:

```
вещ x                           var x: double;
x:= 123.456                     ...
                                x:= 123.456;
```

Вещественное значение нельзя записывать в целочисленную переменную.

В языке Паскаль при выводе на экран вещественных значений по умолчанию используется так называемый **научный** (или **экспоненциальный**) формат, предназначенный для записи как очень больших, так и очень маленьких чисел. Например, программа

```
a:=1;
write(a/3);
```

выведет на экран результат в виде

```
 3.333333E-001
```

что означает $3{,}333333 \cdot 10^{-1} = 0{,}3333333$, т. е. до буквы E указывают значащую часть числа, а после неё — порядок (см. § 29). Для того чтобы вывести вещественное число в «нормальном» виде, используют форматный вывод: после двоеточия записывают общее количество знакомест, которое нужно отвести на число, а затем, снова через двоеточие — количество знаков в дробной части. Та-

кую же форму записи можно использовать и в среде КуМир (начиная с версии 2.0). Например, программа

```
a:=1                          a:=1;
вывод a/3:7:3                 write(a/3:7:3);
```

выводит то же значение в 7 позициях с тремя знаками в дробной части:

```
␣␣0.333
```

Поскольку эта запись занимает 5 позиций (а под неё отведено 7 позиций), перед числом добавляются два пробела, обозначенные знаком «␣».

Стандартные функции

В арифметических выражениях можно использовать стандартные математические функции, например:

`abs(x)` — модуль числа x;
`sqrt(x)` — квадратный корень из числа x (для $x \geq 0$);
`sin(x)` — синус угла x, заданного в радианах;
`cos(x)` — косинус угла x, заданного в радианах;
`exp(x)` — экспонента числа x, т. е. e^x, где $e \approx 2{,}718$ — основание натуральных логарифмов;
`ln(x)` — натуральный логарифм числа x (для $x > 0$).

Последние две функции позволяют выполнить возведение в степень в Паскале, используя равенство $x^y = e^{y \ln x}$.

Существуют функции для перехода от вещественных значений к целым. В языке Паскаль есть функции:

`trunc(x)` — приведение вещественного числа x к целому, отбрасывание дробной части;
`round(x)` — округление вещественного числа x до ближайшего целого.

В школьном алгоритмическом языке есть функция выделения целой части числа: `int(x)`. Нужно иметь в виду, что функция `trunc` в Паскале отбрасывает дробную часть, а функция `int` в школьном алгоритмическом языке находит целую часть по правилам математики (как наибольшее целое число, не превосходящее данное). Поэтому при работе с отрицательными числами они могут давать разный результат:

```
вывод int(-1.5) | = -2      write(trunc(-1.5)); { = -1 }
```

Функция `frac(x)` в языке Паскаль выделяет дробную часть вещественного числа x.

Случайные числа

В некоторых задачах необходимо моделировать случайные события, например результат бросания игрального кубика (на нём может выпасть число от 1 до 6). Как сделать это на компьютере, который по определению «неслучаен», т. е. строго выполняет заданную ему программу?

Случайные числа — это последовательность чисел, в которой невозможно предсказать следующее число, даже зная все предыдущие. Чтобы получить истинно случайные числа, можно, например, бросать игральный кубик или измерять какой-то естественный шумовой сигнал (например, радиошум или электромагнитный сигнал, принятый из космоса). На основе этих данных составлялись и публиковались таблицы случайных чисел, которые использовали в разных областях науки.

Вернёмся к компьютерам. Ставить сложную аппаратуру для измерения естественных шумов или космического излучения на каждый компьютер очень дорого, и повторить эксперимент будет невозможно — завтра все значения будут уже другими. Существующие таблицы слишком малы, когда, скажем, нужно получать 100 случайных чисел каждую секунду. Для хранения больших таблиц требуется много памяти.

Чтобы выйти из положения, математики придумали алгоритмы получения **псевдослучайных** («как бы случайных») **чисел**. Для «постороннего» наблюдателя псевдослучайные числа практически неотличимы от случайных, но они вычисляются по некоторой математической формуле[1]: зная первое число («зерно»), можно по формуле вычислить второе, затем третье и т. п.

В школьном алгоритмическом языке существуют две функции для получения случайных (точнее, псевдослучайных) чисел:

`rand(a,b)` — случайное вещественное число в полуинтервале $[a, b)$;
`irand(a,b)` — случайное целое число на отрезке $[a, b]$.

[1] В стандартные библиотеки языков программирования обычно входит *линейный конгруэнтный генератор* псевдослучайных целых чисел, использующий формулу $X_{k+1} = (aX_k + c) \bmod m$, где X_{k+1} и X_k — следующее и предыдущее псевдослучайные числа; a, c и m — целые числа, подобранные так, чтобы в получаемой цепочке чисел было как можно меньше закономерностей. Например, в трансляторе Borland Delphi использовались значения $a = 134775813$, $c = 1$ и $m = 2^{32}$.

В Паскале есть аналогичные функции:

`random` — случайное вещественное число в полуинтервале [0, 1) (вызов функции без параметров);

`random(N)` — случайное целое число на отрезке [0, $N-1$].

Для того чтобы записать в целую переменную `n` случайное число в диапазоне от 1 до 6 (результат бросания кубика), можно использовать такие операторы:

```
n:=irand(1, 6)          n:=random(6)+1;
```

Вещественное случайное число в полуинтервале от 5 до 12 (не включая 12) получается так:

```
x:=rand(5, 12)          x:=7*random+5;
```

Вопросы и задания

1. Какие типы данных вы знаете?
2. Почему во многих языках программирования есть несколько целочисленных и вещественных типов данных?
3. Чем отличается символьная переменная от строковой?
4. Какие данные записываются в логические переменные? Сколько места в памяти они занимают?
5. Что такое приоритет операций? Зачем он нужен?
6. В каком порядке выполняются операции, если они имеют одинаковый приоритет?
7. Зачем в выражениях используются скобки?
8. Что происходит, если в выражение входят переменные разных типов? Какого типа будет результат?
9. Опишите операции `div` и `mod`. Подумайте, как их можно было бы определить для вещественных чисел.
10. Подумайте, как можно вычислить остаток от деления, если в языке программирования нет для этого специальной операции (`mod`), но есть операция целочисленного деления (`div`).
11. Расскажите о проблеме вычисления остатка в различных языках программирования. Обсудите в классе этот вопрос.
12. Как выполняется операция возведения в степень?
13. Какие стандартные математические функции вы знаете? В каких единицах задаётся аргумент тригонометрических функций?
14. Как выполнить округление (к ближайшему целому) в школьном алгоритмическом языке?
15. Какие числа называют случайными? Зачем они нужны?
16. Как получить «естественное» случайное число? Почему такие числа почти не используются в цифровой технике?
17. Чем отличаются псевдослучайные числа от случайных?
18. Какие функции для получения псевдослучайных чисел вы знаете?

Подготовьте сообщение

а) «Типы данных и переменные в языке Си»
б) «Типы данных и переменные в языке Javascript»
в) «Типы данных и переменные в языке Python»
г) «Вычисление остатка от деления в языках программирования»
д) «Датчики псевдослучайных чисел»

Задачи

1. Найдите в справочной системе или в Интернете диапазон значений для вещественных типов данных.
2. Напишите программу, которая находит сумму, произведение и среднее арифметическое трёх целых чисел, введённых с клавиатуры. Например, при вводе чисел 4, 5 и 7 мы должны получить ответ $4 + 5 + 7 = 16$, $4 \cdot 5 \cdot 7 = 140$, $(4 + 5 + 7)/3 = 5{,}333333$.
3. Напишите программу, которая вводит радиус круга и вычисляет площадь этого круга и длину окружности. На языке Паскаль можно использовать встроенную константу `Pi`, равную числу π.
4. Напишите программу, которая меняет местами значения двух переменных в памяти.

*5. В задаче 4 попробуйте найти решение, которое не использует дополнительные переменные.

6. Напишите программу, которая возводит введённое число в степень 10, используя только четыре операции умножения. Что произойдёт, если ввести большое число, например 78? Попытайтесь объяснить полученный результат.
7. Вычислите значение вещественной переменной c при $a = 2$ и $b = 3$:

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

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

а)
```
c:=mod(a, b)+b
```
```
c:=a mod b + b;
```
б)
```
c:=div(a, b)+a
```
```
c:=a div b + a;
```
в)
```
b:=div(a, b)
c:=div(a, b)
```
```
b:=a div b;
c:=a div b;
```
г)
```
b:=div(a, b)+b
c:=mod(a, b)+a
```
```
b:=a div b + b;
c:=a mod b + a;
```
д)
```
b:=mod(a, b)+4
c:=mod(a, b)+1
```
```
b:=a mod b + 4;
c:=a mod b + 1;
```
е)
```
b:=div(a, b)
c:=mod(a, b+1)
```
```
b:=a div b;
c:=a mod (b+1);
```
ж)
```
b:=mod(a, b)
c:=div(a, b+1)
```
```
b:=a mod b;
c:=a div (b+1);
```

9. Выполните задание 8 при $a = -22$ и $b = 4$. Чем различаются результаты работы программ на школьном алгоритмическом языке и на Паскале?
10. Напишите программу, которая вводит трёхзначное число и разбивает его на цифры. Например, при вводе числа 123 программа должна вывести «1,2,3».
11. Напишите программу, которая вводит четырёхзначное натуральное число и переставляет его первую и последнюю цифры, например, из числа 1234 должно получиться число 4231.
12. Напишите программу, которая вводит четырёхзначное число и «вырезает» из него вторую цифру с начала, например, из числа 1234 должно получиться число 134.
13. Напишите программу, которая вводит четырёхзначное число и «вырезает» из него первую и последнюю цифры, например, из числа 1234 должно получиться число 23.
14. Напишите программу, которая вводит координаты двух точек на числовой оси и выводит расстояние между ними.
15. Напишите программы на обоих языках, которые вводят два вещественных числа (x и y) и вычисляют значение x^y.
*16. Напишите программу на школьном алгоритмическом языке, которая округляет вещественное число до ближайшего целого.
17. Напишите программу, которая вводит два целых числа, a и b ($a < b$) и выводит на экран 5 случайных целых чисел на отрезке $[a, b]$.
18. Напишите программу, которая моделирует бросание двух игральных кубиков: при запуске выводит случайное число в диапазоне от 2 до 12.
19. Напишите программу, которая случайным образом выбирает дежурных: выводит два случайных числа в диапазоне от 1 до N, где N — количество учеников вашего класса. С какой проблемой вы можете столкнуться?
20. Напишите программу, которая вводит два вещественных числа, a и b ($a < b$) и выводит на экран 5 случайных вещественных чисел в полуинтервале $[a, b)$.

§ 57 Ветвления

Условный оператор

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

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

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

```
если a>b то
  M:=a
иначе
  M:=b
все
```

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

Видно, что запись на школьном алгоритмическом языке — это практически точный перевод записи ключевых слов языка Паскаль на русский язык. Обратите внимание, что в языке Паскаль перед ключевым словом **else** (иначе) точка с запятой не ставится.

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

```
M:=a
если b>a то
  M:=b
все
```

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

Здесь использован условный оператор в **неполной форме**, потому что в случае, когда условие ложно, ничего делать не требуется (нет слова **иначе** и операторов после него).

Для того чтобы сделать текст программы более понятным, всё тело условного оператора сдвинуто вправо. Вообще говоря, это не обязательно: в Паскале вообще вся программа может быть записана в одну строку, а в школьном алгоритмическом языке важно только разбиение программы на строки, а отступы игнорируются. Тем не менее запись с отступами значительно повышает читаемость программ, и мы далее будем её использовать[1].

Часто при каком-то условии нужно выполнить сразу несколько действий. Например, в задаче сортировки значений перемен-

[1] В некоторых языках, например в языке Python, отступы обязательны, и все строки одного уровня вложенности должны иметь одинаковые отступы.

ных a и b по возрастанию нужно поменять местами значения этих переменных, если $a > b$:

```
если a>b то
  c:=a
  a:=b
  b:=c
все
```

```
if a>b then begin
  c:=a;
  a:=b;
  b:=c
end;
```

В школьном алгоритмическом языке форма записи совсем не меняется, а на Паскале после ключевого слова **then** нужно записать **составной оператор**, в котором между словами **begin** и **end** может быть сколько угодно команд.

Кроме знаков < и > в условиях можно использовать другие знаки отношений: <= (меньше или равно), >= (больше или равно), = (равно) и <> (не равно, два знака < и > без пробела).

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

```
если a>b то
  вывод 'Андрей старше'
иначе
  если a=b то
    вывод 'Одного возраста'
  иначе
    вывод 'Борис старше'
  все
все
```

```
if a>b then
  writeln('Андрей старше')
else
  if a=b then
    writeln('Одного возраста')
  else
    writeln('Борис старше');
```

Условный оператор, проверяющий равенство, находится внутри блока **иначе (else)**, поэтому он называется **вложенным условным оператором**. Как видно из этого примера, использование вложенных условных операторов позволяет выбрать один из нескольких (а не только из двух) вариантов.

При работе с вложенными условными операторами в языке Паскаль нужно помнить правило: любой блок **else** относится к ближайшему предыдущему оператору **if**, у которого такого блока ещё не было. Например, оператор

```
if a>b then write('A') else if a=b then write('=')
else write('B');
```


может быть записан с выделением структуры так:

```
if a>b then
  write('A')
else
  if a=b then write('=')
         else write('B');
```

Здесь второй блок **else** относится к ближайшему (второму, вложенному) условному оператору, поэтому буква Б будет выведена только тогда, когда оба условия окажутся ложными.

Сложные условия

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

В качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) «И», «ИЛИ» и «НЕ» (см. главу 3). В языке Паскаль они записываются английскими словами: **and**, **or** и **not**.

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

```
если v >= 25 и v <= 40 то
  вывод 'подходит'
иначе
  вывод 'не подходит'
все
```

```
if (v >= 25) and (v <= 40) then
  writeln('подходит')
else
  writeln('не подходит');
```

Обратите внимание, что в Паскале каждое простое условие заключается в скобки. Это связано с тем, что в этом языке отношения имеют более низкий приоритет, чем логические операции, которые в обоих языках выполняются в таком порядке: сначала все операции «НЕ», затем — «И», и в самом конце — «ИЛИ» (во всех случаях слева направо). Для изменения порядка действий используют круглые скобки.

В языке Паскаль есть операция «исключающее ИЛИ» (**xor**), которая имеет такой же приоритет, что и операция «ИЛИ».

Множественный выбор

Условный оператор предназначен, прежде всего, для выбора одного из двух вариантов (простого ветвления). Иногда нужно сделать выбор из нескольких возможных вариантов.

Пусть, например, в переменной m хранится номер месяца, и нужно вывести на экран его русское название. Конечно, в этом случае можно использовать 12 условных операторов:

```
если m=1 то                    if m=1 then
  вывод 'январь' все             write('январь');
если m=2 то                    if m=2 then
  вывод 'февраль' все            write('февраль');
...                            ...
если m=12 то                   if m=12 then
  вывод 'декабрь' все            write('декабрь');
```

Вместо многоточий могут быть записаны аналогичные операторы для остальных значений m. Но во многих языках программирования для подобных случаев есть специальный **оператор выбора**:

```
выбор                          case m of
  при m=1: вывод 'январь'        1:  write('январь');
  при m=2: вывод 'февраль'       2:  write('февраль');
  ...                            ...
  при m=12: вывод 'декабрь'      12: write('декабрь')
иначе вывод 'ошибка'           else  write('ошибка')
все                            end;
```

Кроме очевидных 12 блоков здесь добавлен ещё один, который сигнализирует об ошибочном номере месяца. Он начинается ключевым словом **иначе** (в Паскале — **else**).

В школьном алгоритмическом языке для выбора можно использовать любые условия (а не только равенство). Например, следующий оператор записывает в переменную *sgn* знак значения переменной x:

```
выбор
  при x<0: sgn:=-1
  при x=0: sgn:=0
  при x>0: sgn:=1
все
```

В Паскале можно через запятую указывать список значений, для которых выполняются одинаковые действия. Например, программа для определения количества дней в месяце (для невисокосного года) может быть записана так:

```
case m of
  2: d:=28;
  1,3,5,7,8,10,12: d:=31
  else d:=30
end;
```


Допускаются также диапазоны, в них начальное и конечное значения отделены двумя точками. Следующая программа выводит социальный статус человека в зависимости от возраста:

```
case v of
  0..6:  write('дошкольник');
  7..17: write('школьник')
  else   write('взрослый')
end;
```

Если в каком-то из вариантов нужно выполнить несколько действий, в языке Паскаль используется составной оператор: нужные команды заключаются в **операторные скобки begin** и **end**.

Вопросы и задания

1. Чем отличаются разветвляющиеся алгоритмы от линейных?
2. Как вы думаете, почему не все задачи можно решить с помощью линейных алгоритмов? Приведите примеры таких задач.
3. Как вы думаете, хватит ли линейных алгоритмов и ветвлений для разработки любой программы?
4. Почему нельзя выполнить обмен значений двух переменных в два шага: a:=b; b:=a?
5. Чем различаются условные операторы в полной и неполной формах? Как вы думаете, можно ли обойтись только неполной формой?
6. Какие отношения вы знаете? Как обозначаются отношения «равно» и «не равно»?
7. Что такое сложное условие?
8. Как определяется порядок вычислений в сложном условии? Расскажите об особенностях вычисления значений логических выражений в языке Паскаль.
9. Зачем нужен оператор выбора? Как можно обойтись без него?
10. Расскажите о различиях в операторах выбора в школьном алгоритмическом языке и в Паскале.
11. Как в операторе выбора записать, что нужно делать, если ни один вариант не подошёл?
12. Как в операторе выбора в языке Паскаль выполнить для какого-то варианта несколько операторов?

Подготовьте сообщение

а) «Условные операторы и операторы выбора в языке Си»
б) «Условные операторы в языке Python»

Задачи

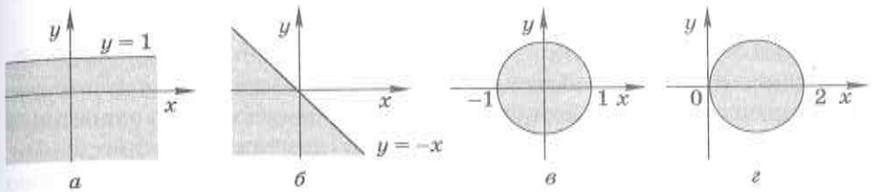
1. Покажите, что приведённая программа не всегда верно определяет максимальное из трёх чисел, записанных в переменные *a*, *b* и *c*:

```
если a>b то M:=a                if a>b then M:=a
иначе      M:=b все             else        M:=b;
если c>b то M:=c                if c>b then M:=c
иначе      M:=b все             else        M:=b;
```

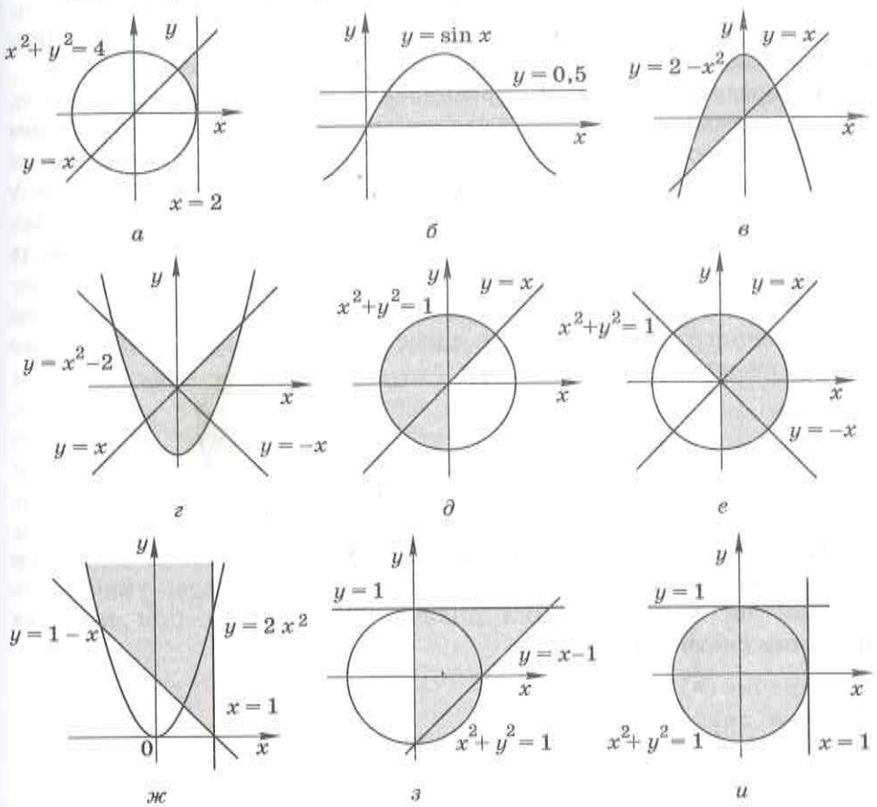
   Приведите контрпример, т. е. значения переменных, при котором в переменной *M* будет получен неверный ответ. Как нужно доработать программу, чтобы она всегда работала правильно?
2. Напишите программу, которая выбирает максимальное и минимальное из пяти введённых чисел.
3. Напишите программу, которая находит среднее из трёх введённых чисел. Предполагается, что все три числа различные.
4. Напишите программу, которая определяет, верно ли, что введённое число трёхзначное.
5. Напишите программу, которая вводит трёхзначное число и десятичную цифру. Определить, входит ли эта цифра в десятичную запись введённого числа.
6. Автомат принимает трёхзначное число, вычисляет сумму двух старших разрядов (сотен и десятков), а также сумму двух младших разрядов (десятков и единиц). Затем эти суммы выводятся на экран в порядке убывания (без пробелов). Напишите программу, которая моделирует работу автомата.
7. Напишите программу, которая вводит четырёхзначное число и определяет, верно ли, что в его десятичной записи цифры стоят в порядке возрастания.
8. Напишите программу, которая вводит четырёхзначное число и определяет, верно ли, что в его десятичной записи ровно две одинаковые цифры.
9. Напишите программу, которая вводит номер месяца и выводит название времени года. Оператор выбора использовать не разрешается. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.
10. Решите предыдущую задачу с помощью оператора выбора.
11. Напишите программу, которая вводит с клавиатуры номер месяца и определяет, сколько дней в этом месяце. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.
12. Напишите программу, которая вводит с клавиатуры номер месяца и день и определяет, сколько дней осталось до Нового года. При вводе неверных данных должно быть выведено сообщение об ошибке.

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

*14. Напишите программу, которая вводит целое число, не превышающее 100, и выводит его прописью, например: 21 → «двадцать один».
15. Напишите программу, которая вводит координаты точки на плоскости и определяет, попала ли эта точка в заштрихованную область.



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



§ 58
Циклические алгоритмы

Как организовать цикл?

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

Простейший цикл, который 10 раз выводит на экран слово «привет», на школьном алгоритмическом языке записывается так:

```
нц 10 раз
  вывод 'привет', нс
кц
```

Здесь **нц** и **кц** — это сокращения от выражений «начало цикла» и «конец цикла».

Подумаем, как можно организовать такой цикл. Вы знаете, что программа выполняется автоматически. При этом на каждом шаге нужно знать, сколько раз уже выполнен цикл и сколько ещё осталось выполнить. Для этого необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла (счётчик шагов). Сначала можно записать в неё ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение ячейки на единицу. На псевдокоде алгоритм можно записать так (здесь и далее операции, входящие в тело цикла, выделяются отступами):

```
счётчик:=0
пока счётчик<10
  вывод 'привет', нс
  увеличить счётчик на 1
```

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

```
счётчик:=10
пока счётчик>0
  вывод 'привет', нс
  уменьшить счётчик на 1
```


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

Циклы с условием

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

Сначала составим алгоритм решения этой задачи. Чтобы считать что-то в программе, нужно использовать переменную, которую называют **счётчиком**. Для подсчёта количества цифр нужно как-то отсекать эти цифры по одной, с начала или с конца, каждый раз увеличивая счётчик. Начальное значение счётчика должно быть равно нулю, так как до выполнения алгоритма ещё не найдено ни одно цифры.

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

Отсечь последнюю цифру проще — достаточно разделить число нацело на 10 (поскольку речь идёт о десятичной системе). Операции отсечения и увеличения счётчика нужно выполнять столько раз, сколько цифр в числе. Как же «поймать» момент, когда цифры кончатся? Несложно понять, что в этом случае результат очередного деления на 10 будет равен нулю, это и говорит о том, что отброшена последняя оставшаяся цифра. Изменение переменной n и счётчика для начального значения 1234 можно записать в виде таблицы на рис. 8.1. Псевдокод выглядит так:

n	*счётчик*
1234	0
123	1
12	2
1	3
0	4

Рис. 8.1

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

Программа на школьном алгоритмическом языке очень похожа на псевдокод, а программа на Паскале почти совпадает с его переводом на английский язык:

```
count:=0
нц пока n>0
  n:=div(n,10)
  count:=count+1
кц
```

```
count:=0;
while n>0 do begin
  n:=n div 10;
  count:=count+1
end;
```

Здесь целочисленная переменная-счётчик имеет имя count. Отметим, что в Паскале после ключевого слова **do** стоит составной оператор, ограниченный ключевыми словами **begin** ... **end**. Если в теле цикла нужно выполнить только один оператор, эти операторные скобки можно не ставить.

Обратите внимание, что проверка условия выполняется в начале цикла. Такой цикл называется **циклом с предусловием** (т. е. с предварительной проверкой условия) или **циклом «пока»**. Если в начальный момент значение переменной n будет нулевое или отрицательное, цикл не выполнится ни одного раза.

В данном случае количество шагов цикла «пока» неизвестно, оно равно количеству цифр введённого числа, т. е. зависит от исходных данных. Цикл этого вида может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k:=0
нц пока k<10
  вывод 'Привет', нс
  k:=k+1
кц
```

```
k:=0;
while k<10 do begin
  writeln('Привет');
  k:=k+1
end;
```

Если условие в заголовке цикла никогда не нарушится, цикл будет работать бесконечно долго. В этом случае говорят, что *программа зациклилась*. Например, если забыть увеличить переменную k в рассмотренном выше примере, программа зациклится:

```
k:=0
нц пока k<10
  вывод 'Привет', нс
кц
```

```
k:=0;
while k<10 do begin
  writeln('Привет')
end;
```


Во многих языках программирования существует **цикл с постусловием**, в котором условие проверяется в конце цикла. Это полезно в том случае, когда нужно обязательно выполнить тело цикла хотя бы один раз. Например, пусть пользователь должен ввести с клавиатуры положительное число. Для того чтобы защитить программу от неверных входных данных, можно использовать цикл с постусловием:

```
нц
  вывод 'Введите n>0: '
  ввод n
кц при n>0
```

```
repeat
  write('Введите n>0: ');
  read(n);
until n>0;
```

Этот цикл закончится тогда, когда станет истинным условие $n>0$ в последней строке, т. е. тогда, когда пользователь введёт допустимое значение. Обратите внимание на особенности этого вида цикла:

- при входе в цикл условие не проверяется, поэтому цикл всегда выполняется хотя бы один раз;
- в последней строке указывают условие окончания цикла (а не условие его продолжения).

Цикл с переменной

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

```
k:=1
n:=2
нц пока k<=10
  вывод n, нс
  n:=n*2
  k:=k+1
кц
```

```
k:=1;
n:=2;
while k<=10 do begin
  writeln(n);
  n:=n*2;
  k:=k+1
end;
```

Вы наверняка заметили, что переменная k используется трижды (см. выделенные блоки): в операторе присваивания начального значения, в условии цикла и в теле цикла (увеличение на 1). Чтобы собрать все действия с ней в один оператор, во многие языки программирования введён особый вид цикла — **цикл с переменной**. В заголовке этого цикла задаются начальное и конечное значения этой переменной, а шаг её изменения по умолчанию равен 1:

```
n:=2
нц для k от 1 до 10
  вывод n, нс
  n:=n*2
кц
```

```
n:=2;
for k:=1 to 10 do begin
  writeln(n);
  n:=n*2
end;
```

Здесь, в отличие от цикла «пока», переменная цикла может быть только целой.

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться на 1. Для этого в школьном алгоритмическом языке добавляется параметр **шаг**, а в Паскале ключевое слово **to** заменяется на **downto** («движение вниз до»). Следующая программа выводит квадраты натуральных чисел от 10 до 1 в порядке убывания:

```
нц для k от 10 до 1 шаг -1
  вывод k*k, нс
кц
```

```
for k:=10 downto 1 do
  writeln(k*k);
```

В школьном алгоритмическом языке шаг изменения переменной цикла может быть любым целым числом, а в Паскале — только 1 или (–1).

Вложенные циклы

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

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

```
нц для n от 2 до 1000
  если число n простое то
    вывод n, нс
  все
кц
```

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

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

```
нц для n от 2 до 1000
  count:=0
  нц для k от 2 до n-1
    если mod(n,k)=0 то
      count:=count+1
    все
  кц
  если count=0 то
    вывод n, нс
  все
кц
```

```
for n:=2 to 1000 do begin
  count:=0;
  for k:=2 to n-1 do
    if n mod k=0 then
      count:=count+1;
  if count=0 then
    writeln(n);
end;
```

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

```
count:=0
k:=2
нц пока k*k<=n
  если mod(n,k)=0 то
    count:=count+1
  все
  k:=k+1
кц
```

```
count:=0;
k:=2;
while k*k<=n do begin
  if n mod k=0 then
    count:=count+1;
  k:=k+1
end;
```

Чтобы ещё ускорить работу цикла, заметим, что когда найден хотя бы один делитель, число уже заведомо составное, и искать другие делители в данной задаче не требуется. Поэтому можно закончить цикл. Для этого в условие работы цикла добавляется условие **mod**(n,k)<>0 (в языке Паскаль n **mod** k <> 0), связанное с имеющимся условием с помощью операции «И»; при этом можно обойтись без переменной *count*.

```
k:=2
нц пока k*k<=n и mod(n,k)<>0
  k:=k+1
кц
если  k*k>n то
  вывод n, нс
все
```

```
k:=2
while (k*k<=n) and
      (n mod k<>0) do
       k:=k+1;
if k*k>n then
  writeln (n);
```

После выхода из цикла проверяется, какое условие было нарушено. Если k*k>n, то число n — простое. В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
нц для i от 1 до 4
  нц для k от 1 до i
  ...
  кц
кц
```

```
for i:=1 to 4 do begin
  for k:=1 to i do begin
  ...
  end
end;
```

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

Вопросы и задания

1. Что такое цикл?
2. Сравните цикл с переменной и цикл с условием. Какие преимущества и недостатки есть у каждого из них?
3. Что означает выражение «цикл с предусловием»?
4. В каком случае цикл с предусловием не выполняется ни разу?
5. В каком случае программа, содержащая цикл с условием, может зациклиться? Приведите пример такой программы.
6. В каком случае цикл с переменной не выполняется ни разу?
7. Верно ли, что любой цикл с переменной можно заменить циклом с условием? Верно ли обратное утверждение? Ответ обоснуйте.
8. В каком случае можно заменить цикл с условием на цикл с переменной?
9. Как будет работать приведённая в параграфе программа, которая считает количество цифр введённого числа, при вводе отрицательного числа? Если вы считаете, что она работает неправильно, укажите, как её нужно доработать.

Подготовьте сообщение

а) «Операторы цикла в языке Си»

б) «Операторы цикла в языке Python»

Задачи

1. Напишите программу, которая вводит два целых числа и находит их произведение, не используя операцию умножения. Учтите, что числа могут быть отрицательными.
2. Напишите программу, которая вводит натуральное число N и находит сумму всех натуральных чисел от 1 до N. Используйте сначала цикл с условием, а потом — цикл с переменной.
3. Напишите программу, которая вводит натуральное число N и выводит первые N чётных натуральных чисел.
4. Напишите программу, которая вводит натуральные числа a и b, и выводит квадраты натуральных чисел в диапазоне от a до b. Например, если ввести 4 и 5, программа должна вывести:
   *4*4=16*
   *5*5=25*
5. Напишите программу, которая вводит натуральные числа a и b, и выводит сумму квадратов натуральных чисел в диапазоне от a до b.
6. Напишите программу, которая вводит натуральное число N и выводит на экран N псевдослучайных чисел. Запустите её несколько раз, объясните результаты опыта.
7. Напишите программу, которая строит последовательность из N случайных чисел на отрезке от 0 до 1 и определяет, сколько из них попадает в полуинтервалы [0; 0,25), [0,25; 0,5), [0,5; 0,75) и [0,75; 1). Сравните результаты, полученные при N = 10, 100, 1000, 10000.
8. Найдите все пятизначные числа, которые при делении на 133 дают в остатке 125, а при делении на 134 дают в остатке 111.
9. Напишите программу, которая вводит натуральное число N и выводит на экран все натуральные числа, не превосходящие N и делящиеся на каждую из своих цифр.
10. *Числа Армстронга.* Натуральное число называется числом Армстронга, если сумма цифр числа, возведённых в N-ю степень, где N — количество цифр в числе, равна самому числу. Например: $153 = 1^3 + 5^3 + 3^3$. Найдите все трёхзначные и четырёхзначные числа Армстронга.
11. *Автоморфные числа.* Натуральное число называется автоморфным, если оно равно последним цифрам своего квадрата. Например: $25^2 = 625$. Напишите программу, которая вводит натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.
12. Напишите программу, которая считает количество чётных цифр введённого числа.
13. Напишите программу, которая считает сумму цифр введённого числа.
14. Напишите программу, которая определяет, верно ли, что введённое число содержит две одинаковые цифры, стоящие рядом (как, например, 221).

15. Напишите программу, которая переставляет первую и последнюю цифры введённого числа, например, из числа 12345 должно получиться 52341.
16. Напишите программу, которая определяет, верно ли, что введённое число состоит из одинаковых цифр (как, например, 222).
*17. Напишите программу, которая определяет, верно ли, что введённое число содержит по крайней мере две одинаковые цифры, возможно, не стоящие рядом (как, например, 212).
18. Используя сначала цикл с условием, а потом — цикл с переменной, напишите программу, которая выводит на экран чётные степени числа 2 от 2^{10} до 2^2 в порядке убывания.
*19. Напишите программу, которая определяет, верно ли, что введённое число содержит ровно три одинаковые цифры, а все остальные встречаются только по одному разу (например, как в числе 77237).
20. *Алгоритм Евклида* для вычисления наибольшего общего делителя (НОД) двух натуральных чисел формулируется так: нужно заменять большее число на разность большего и меньшего до тех пор, пока одно из них не станет равно нулю; тогда второе и есть НОД. Напишите программу, которая реализует этот алгоритм. Какой цикл тут нужно использовать?
21. Напишите программу, использующую *модифицированный алгоритм Евклида*: нужно заменять большее число на остаток от деления большего на меньшее до тех пор, пока этот остаток не станет равен нулю; тогда второе число и есть НОД.
22. Добавьте в решение двух предыдущих задач вычисление количества шагов цикла. Заполните таблицу (в строки «шаги-1» и «шаги-2» записывается количество шагов для двух версий алгоритма Евклида).

a	64168	358853	6365133	17905514	549868978
b	82678	691042	11494962	23108855	298294835
НОД(a, b)					
шаги-1					
шаги-2					

23. Напишите программу, которая вводит последовательность целых чисел, заканчивающуюся нулём, и определяет количество чётных положительных чисел в этой последовательности.
24. Напишите программу, которая вводит последовательность целых чисел, заканчивающуюся нулём, и определяет, является ли она возрастающей (неубывающей).
25. Напишите программу, которая вводит последовательность целых чисел, заканчивающуюся нулём, и определяет, верно ли, что знаки элементов этой последовательности чередуются.
26. Напишите программу, которая вводит последовательность целых чисел, заканчивающуюся нулём, и определяет, сколько из этих чисел удовлетворяет условию: сумма значений цифр десятичной записи числа равна 10.
*27. Автомат принимает трёхзначное число, вычисляет сумму двух старших разрядов (сотен и десятков), а также сумму двух младших разрядов (десятков и единиц). Затем эти суммы выводятся на экран в порядке убывания (без пробелов). Напишите программу, которая по известному результату работы автомата строит все возможные варианты входных чисел.
28. Напишите программу, которая вводит с клавиатуры 10 чисел и вычисляет их сумму и произведение.
29. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводится сумма и произведение введённых чисел (не считая 0).
30. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводятся минимальное и максимальное из введённых чисел (не считая 0).
31. Напишите программу, которая вводит с клавиатуры натуральное число N и определяет его *факториал*, т. е. произведение натуральных чисел от 1 до N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$. Что будет, если ввести большое значение N (например, 20)?
32. Напишите программу, которая вводит натуральные числа A и N и вычисляет A^N без использования операции возведения в степень.
33. Напишите программу, которая выводит на экран в столбик все цифры числа, начиная с первой.
34. Ряд чисел Фибоначчи задаётся следующим образом: первые два числа равны 1 ($F_1 = F_2 = 1$), а каждое следующее равно сумме двух предыдущих: $F_n = F_{n-1} + F_{n-2}$. Напишите программу, которая вводит натуральное число N и выводит на экран первые N чисел Фибоначчи.
35. Напишите программу, которая вводит натуральные числа a и b и выводит все простые числа в диапазоне от a до b.
36. *Совершенным* называется число, равное сумме всех своих делителей, меньших его самого (например, число 6 = 1 + 2 + 3). Напишите программу, которая вводит натуральное число N и определяет, является ли число N совершенным.
37. Напишите программу, которая вводит натуральное число N и находит все совершенные числа в диапазоне от 1 до N.
38. В магазине продаётся мастика в ящиках по 15, 17, 21 кг. Как купить ровно 185 кг мастики, не вскрывая ящики? Сколькими способами можно это сделать?
*39. Ввести натуральное число N и вывести значение числа $1/N$, выделив период дроби. Например: 1/2 = 0,5 или 1/7 = 0,(142857).
*40. В телевикторине участнику предлагают выбрать один из трёх закрытых чёрных ящиков, причём известно, что в одном из них приз, а в двух других пусто. После этого ведущий открывает один пустой

ящик (но не тот, который выбрал участник) и предлагает заново сделать выбор, но уже между двумя оставшимися ящиками. Используя псевдослучайные числа, выполните моделирование 1000 раундов этой игры и определите, как следует поступить участнику, чтобы с большей вероятностью получить приз: выбрать тот же ящик, что и в начале игры, или другой[1].

§ 59 Процедуры

Что такое процедура?

Предположим, что в нескольких местах программы требуется выводить на экран сообщение об ошибке: `Ошибка программы`. Это можно сделать, например, так:

```
вывод 'Ошибка программы'
```

```
write('Ошибка программы');
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести сообщение об ошибке. Но тут есть две сложности. Во-первых, при этом строка-сообщение будет храниться в памяти много раз. Во-вторых, если мы задумаем поменять текст сообщения, нужно будет искать эти операторы вывода по всей программе. Для таких случаев в языках программирования предусмотрены **процедуры — вспомогательные алгоритмы**, которые выполняют некоторые действия.

```
алг С процедурой
нач
  цел n
  ввод n
  если n<0 то Error все
  ...
кон
```

```
алг Error
нач
  вывод 'Ошибка программы'
кон
```

```
program withProc;
var n: integer;
procedure Error;
begin
  writeln('Ошибка программы')
end;
begin
  read(n);
  if n<0 then Error;
  ...
end.
```

Обратим внимание на разницу оформления программы с процедурой в школьном алгоритмическом языке и в Паскале. В школьном алгоритмическом языке процедура оформляется точно так же, как и основной алгоритм, но размещается после основной программы.

[1] Эта задача известна как «парадокс Монти Холла», потому что её решение противоречит интуиции и «здравому смыслу».

В Паскале процедура начинается с ключевого слова **procedure**, тело процедуры начинается с ключевого слова **begin** и заканчивается ключевым словом **end** с точкой с запятой. Процедура располагается после блока объявления переменных, но выше основной программы.

Фактически мы ввели в язык программирования новую команду `Error`, которая была расшифрована прямо в теле программы. Для того чтобы процедура заработала, в основной программе (или в другой процедуре) необходимо ее **вызвать** по имени.

Как мы видели, использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Кроме того, иногда большую программу разбивают на несколько процедур для удобства, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной.

Процедура с параметрами

Процедура `Error` при каждом вызове делает одно и то же. Более интересны процедуры, умеющие решать целый класс задач в зависимости от переданных им данных, которые называются **аргументами**. Внутри процедуры эти данные обозначаются именами и называются **параметрами**.

Предположим, что в программе требуется многократно выводить на экран запись целого числа (0..255) в 8-битном двоичном коде. Старшая цифра в такой записи — это частное от деления числа на 128. Далее возьмём остаток от этого деления и разделим на 64 — получается вторая цифра и т. д. Алгоритм, решающий эту задачу для значения переменной n, можно записать так:

```
k:=128
нц пока k>0
  вывод div(n,k)
  n:= mod(n,k)
  k:= div(k,2)
кц
```

```
k:=128;
while k>0 do begin
  write(n div k);
  n:= n mod k;
  k:= k div 2
end;
```

Писать такой цикл каждый раз, когда нужно вывести двоичное число, очень утомительно. Кроме того, легко сделать ошибку

или опечатку, которую будет сложно найти. Поэтому лучше оформить этот вспомогательный алгоритм в виде процедуры. Но этой процедуре нужно передать аргумент — число для перевода в двоичную систему. Программа получается такой:

```
алг Двоичный код
нач
  printBin(99)
кон
алг printBin(цел n0)
нач
  цел n, k
  n:= n0
  k:= 128
  нц пока k>0
    вывод div(n,k)
    n:= mod(n,k)
    k:= div(k,2)
  кц
кон
```

```
program binCode;
procedure printBin(n: integer);
var k: integer;
begin
  k:= 128;
  while k>0 do begin
    write(n div k);
    n:= n mod k;
    k:= k div 2
  end
end;
begin
  printBin(99)
end.
```

Основная программа содержит всего одну команду — вызов процедуры printBin для значения 99 (это аргумент функции). Внутри процедуры это значение называется **параметром**, к нему обращаются по имени. В заголовке процедуры в скобках записывают тип и внутреннее имя параметра.

В процедуре объявлена **локальная** (внутренняя) **переменная** *k* — она известна только внутри этой процедуры. Обратите внимание, что в школьном алгоритмическом языке локальные переменные объявляются после ключевого слова **нач**, а в языке Паскаль — до слова **begin**.

В школьном алгоритмическом языке запрещено изменять параметры процедуры внутри процедуры, поэтому мы назвали параметр *n*0, а в начале процедуры скопировали его значение в переменную *n*.

Параметров может быть несколько, в этом случае они перечисляются в заголовке процедуры через запятую (в школьном алгоритмическом языке) или точку с запятой (в Паскале). Например, процедуру, которая выводит на экран среднее арифметическое двух целых чисел, можно записать так:


```
алг printSred(цел a,
              цел b)
нач
  вывод (a+b)/2
кон
```

```
procedure printSred
                (a: integer;
                 b: integer);
begin
  write((a+b)/2)
end.
```

Если несколько параметров одного типа стоят в списке один за другим, их можно определить списком:

```
алг printSred(цел a, b)
нач
  вывод (a+b)/2
кон
```

```
procedure printSred(a, b: integer);
begin
  write((a+b)/2)
end.
```

Изменяемые параметры

Напишем процедуру, которая меняет местами значения двух переменных. Проще всего для этого использовать третью переменную (пока напишем программу только на Паскале):

```
program Exchange;
var x, y: integer;
procedure Swap(a, b: integer);
var c: integer;
begin
  c:=a; a:=b; b:=c
end;
begin
  x:=2; y:=3;
  Swap(x, y);
  write(x, ' ', y)
end.
```

После запуска этой программы обнаружится, что значения переменных *x* и *y* остались прежними: на экран будет выведено: 2 3. Дело в том, что эта процедура работает с *копиями* переданных ей значений. Это значит, что процедура Swap создаёт в памяти временные локальные переменные с именами *a* и *b* и копирует в них переданные значения переменных *x* и *y* основной программы. Поэтому и все перестановки в нашей программе были сдела-

ны именно с копиями, а значения переменных x и y основной программы не изменились. Такая передача данных в процедуру называется **передачей по значению**.

Чтобы решить проблему, нужно явно сказать, чтобы процедура работала с теми же ячейками памяти, что и основная программа. Для этого в Паскале в заголовке процедуры перед именем изменяемого параметра пишут ключевое слово **var**:

```
procedure Swap( var a, b: integer);
```

Теперь процедура решает поставленную задачу: на выходе мы увидим: 3 2, что и требовалось. В подобных случаях говорят, что данные **передаются по ссылке**, а не по значению. Это означает, что фактически в процедуру передаётся адрес переменной и можно изменять значение этой переменной, записывая новые данные по этому адресу.

Теперь при вызове процедуре Swap можно передавать только переменные, но не константы (постоянные) и не арифметические выражения. Например, вызовы Swap(2, 3) и Swap(x, y+3) противоречат правилам языка программирования, и программа выполняться не будет.

В школьном алгоритмическом языке все параметры делятся на **аргументы** (исходные данные, обозначаются **арг**) и **результаты** (ключевое слово **рез**, эти значения процедура передаёт вызывающей программе). По умолчанию (если не указано иначе) все параметры считаются аргументами. Поэтому два следующих заголовка равносильны:

```
алг Swap(цел a, b)
```

и

```
алг Swap( арг цел a, b)
```

В нашем случае параметры процедуры одновременно являются и аргументами, и результатами, поэтому их нужно объявлять с помощью ключевого слова **аргрез**. Приведём полную программу:

```
алг Обмен
нач
  цел x=2, y=3
  Swap(x, y)
  вывод x, ' ', y
кон
```



```
алг Swap(аргрез цел a, b)
нач
  цел c
  c:=a; a:=b; b:=c
кон
```

Вопросы и задания

1. Что такое процедуры? В чём смысл их использования?
2. Как оформляются процедуры в школьном алгоритмическом языке и в Паскале?
3. Достаточно ли включить процедуру в текст программы, чтобы она «сработала»?
4. Что такое параметры? Зачем они используются?
5. Какие переменные называются локальными? Где они объявляются?
6. Как оформляются процедуры, имеющие несколько параметров?
7. Что такое изменяемые параметры? Зачем они используются?
8. Как в заголовке процедуры отличить изменяемый параметр от неизменяемого? Приведите примеры.
9. Какие типы параметров выделяются в школьном алгоритмическом языке? Как они объявляются?

Подготовьте сообщение

а) «Процедуры в языке Си»
б) «Процедуры в языке Python»

Задачи

1. Напишите процедуру, которая выводит на экран запись числа, меньшего, чем 8^{10}, в виде 10 знаков в восьмеричной системе счисления.
2. Напишите процедуру, которая выводит на экран запись числа, меньшего, чем $16^4 = 65\,536$, в виде 4 знаков в шестнадцатеричной системе счисления.
3. Напишите процедуру, которая принимает параметр — натуральное число N и выводит на экран квадрат из звёздочек со стороной N.
4. Напишите процедуру, которая принимает числовой параметр — возраст человека в годах и выводит этот возраст со словом «год», «года» или «лет». Например, 21 год, 22 года, 12 лет.

5. Напишите процедуру, которая выводит переданное ей число прописью. Например: 21 → двадцать один.
6. Напишите процедуру, которая принимает параметр — натуральное число N и выводит первые N чисел Фибоначчи (см. задачу 34 к § 58).
7. Напишите процедуру, которая определяет, верно ли, что переданное ей число — простое. (Используйте изменяемые параметры.)
8. Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с последней.
9. Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с первой.
10. Напишите процедуру, которая выводит на экран все делители переданного ей числа (в одну строчку).
11. Напишите процедуру, которая принимает параметр — натуральное число N — и выводит на экран линию из N символов '-'.

*12. Напишите процедуру, которая выводит на экран запись переданного ей числа в римской системе счисления.

§ 60 Функции

Пример функции

С функциями вы уже знакомы, потому что наверняка применяли стандартные функции языка программирования (например, abs, sin, cos). **Функция**, как и процедура, — это **вспомогательный алгоритм**, который может принимать **аргументы**. Но, в отличие от процедуры, функция всегда *возвращает* **значение-результат**. Результатом может быть число, символ, символьная строка или данные другого типа.

Составим функцию, которая вычисляет сумму цифр числа. Будем использовать следующий алгоритм (предполагается, что число записано в переменной n):

```
сумма:=0
нц пока n<>0
  сумма:=сумма+mod(n,10)
  n:=div(n,10)
кц
```


Чтобы получить последнюю цифру числа (которая добавляется к сумме), нужно взять остаток от деления числа на 10. Затем последняя цифра отсекается, и мы переходим к следующей цифре. Цикл продолжается до тех пор, пока значение n не станет равно нулю.

Как указать в программе, чему равно значение функции? Для этого часто используют такой приём: значение функции записывается в специальную переменную. В школьном алгоритмическом языке имя этой переменной — **знач**, а в Паскале оно совпадает с именем функции (во многих версиях Паскаля можно использовать вместо этого встроенную переменную *Result*):

```
алг Сумма цифр
нач
 вывод sumDigits(12345)
кон
алг цел sumDigits(цел n0)
нач
  цел sum=0, n
  n:=n0
  нц пока n<>0
    sum:=sum+mod(n,10)
    n:=div(n,10)
  кц
  знач:=sum
кон
```

```
program Sum;
function sumDigits(n: integer):
                          integer;
var sum: integer;
begin
  sum:=0;
  while n<>0 do begin
    sum:=sum+n mod 10;
    n:=n div 10
  end;
  sumDigits:=sum
end;
begin
  write(sumDigits(12345))
end.
```

Обратим внимание на особенности записи: тип возвращаемого значения указывается в заголовке функции (в школьном алгоритмическом языке — перед именем функции, в Паскале — в конце заголовка через двоеточие). Так же как и в процедурах, в функциях можно объявлять и использовать локальные переменные. Они входят в «зону видимости» только этой функции, для всех остальных функций и процедур они недоступны.

Функции, созданные в программе таким образом, применяются точно так же, как и стандартные функции. Их можно вызывать везде, где может использоваться выражение того типа, кото-

рый возвращает функция. Приведём несколько примеров вызова функций на школьном алгоритмическом языке:

```
x:=2*sumDigits(n+5)
z:=sumDigits(k) + sumDigits(m)
если mod(sumDigits(n),2)=0 то
  вывод 'Сумма цифр чётная', нс
  вывод 'Она равна ', sumDigits(n)
все
```

Логические функции

Достаточно часто применяют специальный тип функций, которые возвращают **логическое значение** (да или нет, True или False). Иначе говоря, такая **логическая функция** отвечает на вопрос «Да или нет?» и возвращает 1 бит информации.

Вернёмся к задаче, которую мы уже рассматривали: вывести на экран все простые числа в диапазоне от 2 до 1000. Алгоритм определения простоты числа оформим в виде функции. При этом его можно легко вызвать из любой точки программы.

Запишем основную программу на псевдокоде:

```
алг Простые числа
нач
  цел i
  нц для i от 2 до 100
    если i - простое то
      вывод i, нс
    все
  кц
кон
```

```
program PrimeNum;
var i: integer;
begin
  for i:=2 to 100 do
    if i - простое then
      writeln(i)
end.
```

Предположим, что у нас уже есть логическая функция isPrime, которая определяет простоту переданного ей числа и возвращает да (в Паскале — True), если число простое, и нет (False) в противном случае. Такую функцию можно использовать вместо выделенного блока алгоритма:

```
если isPrime(i) то
  вывод i, нс
кц
```

```
if isPrime(i) then
  writeln(i);
```


Остаётся написать саму функцию isPrime. Будем использовать уже известный алгоритм: если число n в интервале от 2 до $\sqrt{n}$ не имеет ни одного делителя, то оно простое[1]:

```
алг лог isPrime(цел n)
нач
  цел k
  k:=2
  нц пока k*k<=n и mod(n,k)<>0
    k:=k+1
  кц
  знач:=(k*k>n)
кон
```

```
function isPrime(n: integer):
                          boolean;
var k: integer;
begin
  k:=2;
  while (k*k<=n) and
        (n mod k<>0) do
    k:=k+1
  isPrime:=(k*k>n)
end;
```

Эта функция возвращает логическое значение, на это указывает ключевое слово **лог** (в Паскале — **boolean**). Значение функции определяется как

```
знач:=(count=0)
```

Это оператор присваивания. Слева от знака «:=» записана **логическая переменная**, а справа — **логическое выражение (условие)**. Если это выражение истинно, то в переменную знач записывается значение да, иначе — значение нет.

Логические функции можно использовать так же, как и любые условия: в условных операторах и циклах с условием. Например, такой цикл останавливается на первом введённом составном числе:

```
ввод n
нц пока isPrime(n)
  вывод 'простое число', нс
  ввод n
кц
```

```
read(n);
while isPrime(n) do begin
  writeln('простое число');
  read(n)
end;
```

[1] Эту программу можно ещё немного усовершенствовать: после числа 2 имеет смысл проверять только нечётные делители, увеличивая на каждом шаге значение k сразу на 2.

Вопросы и задания

1. Что такое функция? Чем она отличается от процедуры?
2. Как оформляются функции в тексте программы (сравните школьный алгоритмический язык и Паскаль)?
3. Как по тексту программы определить, какое значение возвращает функция? Приведите пример.
4. Какие функции называются логическими? Зачем они нужны?

Подготовьте сообщение

а) «Функции в языке Си»
б) «Функции в языке Python»

Задачи

1. Напишите функцию, которая вычисляет максимальное из трёх чисел.
2. На соревнованиях выступление спортсмена оценивают 5 экспертов, каждый из них выставляет оценку в баллах (целое число). Для получения итоговой оценки лучшая и худшая из оценок экспертов отбрасываются, а для оставшихся трёх находится среднее арифметическое. Напишите функцию, которая принимает 5 оценок экспертов и возвращает итоговую оценку выступления спортсмена.
3. Напишите функцию, которая вычисляет количество цифр числа.
4. Напишите функцию, которая вычисляет наибольший общий делитель двух чисел.
5. Напишите функцию, которая вычисляет наименьшее общее кратное двух чисел.
6. Напишите функцию, которая «разворачивает» десятичную запись числа наоборот, например из 123 получается 321, а из 3210 — 0123.
7. Напишите функцию, которая моделирует бросание двух игральных кубиков (на каждом может выпасть от 1 до 6 очков). (Используйте генератор псевдослучайных чисел.)
8. Напишите функцию, которая вычисляет факториал натурального числа N.
9. Напишите функцию, которая вычисляет N-е число Фибоначчи.
10. *Дружественные числа* — это два натуральных числа, таких что сумма всех делителей одного числа (меньших самого этого числа) равна другому числу, и наоборот. Найдите все пары дружественных чисел, каждое из которых меньше 10 000. Используйте функцию, которая вычисляет сумму делителей числа.
11. Напишите программу, которая вводит натуральное число N и находит все числа на отрезке [0, N], сумма цифр которых не меняется при умножении числа на 2, 3, 4, 5, 6, 7, 8 и 9 (например, число 9). Используйте функцию для вычисления суммы цифр числа.

12. Доработайте функцию `isPrime` так, чтобы она возвращала значение `нет` (`False`) для всех чисел, меньших, чем 2.
13. Напишите логическую функцию, которая определяет, верно ли, что число N — совершенное, т. е. равно сумме своих делителей, меньших его самого.
14. Простое число называется *гиперпростым*, если любое число, получающееся из него отбрасыванием нескольких последних цифр, тоже является простым. Например, число 733 — гиперпростое, так как и оно само, и числа 73 и 7 — простые. Напишите логическую функцию, которая определяет, верно ли, что число N — гиперпростое. Используйте уже готовую функцию `isPrime`.

§ 61 Рекурсия

Что такое рекурсия?

Вспомним определение натуральных чисел. Оно состоит из двух частей:

1) 1 — натуральное число;
2) если n — натуральное число, то $n + 1$ — тоже натуральное число.

Вторая часть этого определения в математике называется **индуктивной**: натуральное число определяется через другое натуральное число, и таким образом определяется всё множество натуральных чисел. В программировании этот приём называют рекурсией.

Рекурсия — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

Первая часть в определении натуральных чисел — это и есть тот самый базовый случай. Если убрать первую часть из определения, оно будет неполно: вторая часть даёт только метод перехода к следующему значению, но не даёт никакой «зацепки», не отвечает на вопрос «откуда начать».

Похожим образом задаются **числа Фибоначчи**: первые два числа равны 1, а каждое из следующих чисел равно сумме двух предыдущих:

1) $F_1 = F_2 = 1$;
2) $F_n = F_{n-1} + F_{n-2}$ для $n > 2$.

Популярные примеры рекурсивных объектов — **фракталы**. Так в математике называют геометрические фигуры, обладающие **самоподобием**. Это значит, что они составлены из фигур меньшего размера, каждая из которых подобна целой фигуре. На рисунке 8.2 показан **треугольник Серпинского** — один из первых фракталов, который предложил в 1915 г. польский математик В. Серпинский.

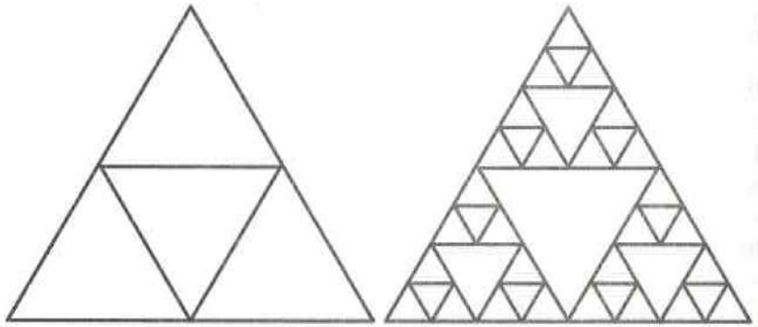


Рис. 8.2

Равносторонний треугольник делится на 4 равных треугольника меньшего размера (см. рис. 8.2, слева), затем каждый из полученных треугольников, кроме центрального, снова делится на 4 ещё более мелких треугольника и т. д. На рисунке 8.2 справа показан треугольник Серпинского с тремя уровнями деления.

Ханойские башни

Согласно легенде, конец света наступит тогда, когда монахи Великого храма города Бенарас смогут переложить 64 диска разного диаметра с одного стержня на другой. Вначале все диски нанизаны на первый стержень. За один раз можно перекладывать только один диск, причём разрешается класть только меньший диск на больший. Есть также и третий стержень, который можно использовать в качестве вспомогательного (рис. 8.3).

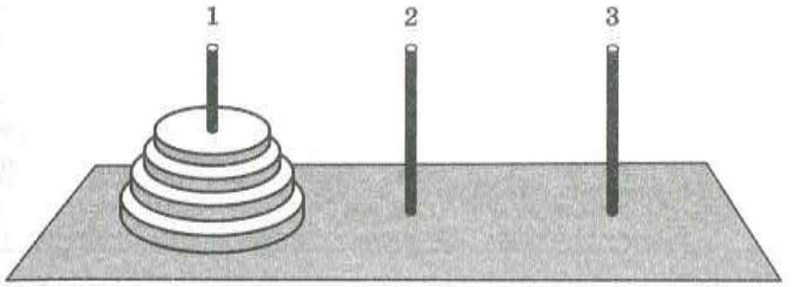


Рис. 8.3

Решить задачу для 2, 3 и даже 4 дисков довольно просто. Проблема в том, чтобы составить алгоритм для любого числа дисков.

Пусть нужно перенести n дисков со стержня 1 на стержень 3. Представим себе, что мы как-то смогли переместить $n - 1$ дисков на вспомогательный стержень 2. Тогда остаётся перенести самый большой диск на стержень 3, а затем $n - 1$ меньших дисков со вспомогательного стержня 2 на стержень 3, и задача будет решена. Получается такой псевдокод:

```
перенести (n-1, 1, 2)
1 -> 3
перенести (n-1, 2, 3)
```

Здесь запись `1 → 3` обозначает: перенести диск со стержня 1 на стержень 3. Процедура `перенести` (которую нам предстоит написать) принимает три аргумента: число дисков, номера начального и конечного стержней. Таким образом, мы свели задачу переноса n дисков к двум задача переноса $n - 1$ дисков и одному элементарному действию — переносу одного диска. Заметим, что при известных номерах начального и конечного стержней легко рассчитать номер вспомогательного, так как сумма номеров равна 1 + 2 + 3 = 6. Получается такая (пока не совсем верная) процедура:

```
алг Hanoi(цел n, k, m)
нач
  цел p
  p:=6-k-m
  Hanoi(n-1, k, p)
  вывод k, ' -> ', m, нс
  Hanoi(n-1, p, m)
кон
```

Здесь переменная p обозначает номер вспомогательного стержня. Эта процедура вызывает сама себя, но с другими аргументами. Такая процедура называется рекурсивной.

Рекурсивная процедура (функция) — это процедура (функция), которая вызывает сама себя напрямую или через другие процедуры и функции.

Основная программа для решения задачи, например с четырьмя дисками, будет содержать всего одну строку (перенести 4 диска со стержня 1 на стержень 3):

```
Hanoi(4, 1, 3)
```

Если вы попробуете запустить эту программу, она зациклится, т. е. будет работать бесконечно долго. В чём же дело? Вспомните, что определение рекурсивного объекта состоит из двух частей. В первой части определяются базовые объекты (например, первое натуральное число), именно эта часть «отвечает» за остановку рекурсии в программе. Пока мы не определили такое условие останова, процедура будет бесконечно вызывать саму себя (это можно проверить, выполняя программу по шагам). Когда же остановить рекурсию?

Очевидно, что если нужно переложить 0 дисков, задача уже решена, ничего перекладывать не надо. Это и есть условие окончания рекурсии: если значение параметра n, переданное процедуре, равно 0, нужно выйти из процедуры без каких-либо действий. Для этого в школьном алгоритмическом языке используется оператор `выход` (в Паскале — оператор `exit`). Правильная версия процедуры выглядит так:


```
алг Hanoi(цел n, k, m)
нач
  если n=0 то выход все
  цел p
  p:=6-k-m
  Hanoi(n-1, k, p)
  вывод k, ' -> ', m, нс
  Hanoi(n-1, p, m)
кон
```

```
procedure Hanoi(n, k, m: integer);
var p: integer;
begin
  if n=0 then exit;
  p:=6-k-m;
  Hanoi(n-1, k, p);
  writeln(k, ' -> ', m);
  Hanoi(n-1, p, m)
end;
```

Чтобы переложить N дисков, нужно выполнить $2^N - 1$ перекладываний. Для $N = 64$ это число равно 18 446 744 073 709 551 615. Если бы монахи, работая день и ночь, каждую секунду перемещали один диск, им бы потребовалось 580 миллиардов лет.

Примеры

Пример 1. Составим процедуру, которая переводит натуральное число в двоичную систему. Мы уже занимались вариантом этой задачи, где требовалось вывести 8-битную запись числа из диапазона 0..255, сохранив лидирующие нули. Теперь усложним задачу: лидирующие нули выводить не нужно, а натуральное число может быть любым (в пределах допустимого диапазона для выбранного типа данных).

Стандартный алгоритм перевода числа в двоичную систему можно записать, например, так:

```
нц пока n<>0
  вывод mod(n,2)
  n:=div(n,2)
кц
```

Проблема в том, что двоичное число выводится «задом наперёд», т. е. первым будет выведен последний разряд. Как «перевернуть» число?

Есть разные способы решения этой задачи, которые сводятся к тому, чтобы запоминать остатки от деления (например, в символьной строке) и затем, когда результат будет полностью получен, вывести его на экран.

Однако можно применить красивый подход, использующий рекурсию. Идея такова: чтобы вывести двоичную запись числа `n`, нужно сначала вывести двоичную запись числа `div(n, 2)`, а затем последнюю цифру — `mod(n, 2)`. Если число-параметр равно

нулю, нужно выйти из процедуры. Такой алгоритм очень просто программируется:

```
алг printBin(цел n)
нач
  если n=0 то выход все
  printBin(div(n, 2))
  вывод mod(n,2)
кон
```

```
procedure printBin(n: integer);
begin
  if n=0 then exit;
  printBin(n div 2);
  write(n mod 2)
end;
```

Конечно, решить эту задачу можно было и с помощью цикла. Поэтому можно сделать важный вывод: *рекурсия заменяет цикл*. При этом программа во многих случаях становится более понятной.

Пример 2. Составим функцию, которая вычисляет сумму цифр числа. Будем рассуждать так: сумма цифр числа n равна значению последней цифры плюс сумма цифр числа div(n, 10). Сумма цифр однозначного числа равна самому этому числу, это условие окончания рекурсии. Получаем следующую функцию:

```
алг цел sumDig(цел n)
нач
  знач:=mod(n,10)
  если n>=10 то
    знач:=знач+sumDig(div(n,10))
  все
кон
```

```
function sumDig
            (n: integer):
               integer;
var sum: integer;
begin
  sum:=n mod 10;
  if n>=10 then
    sum:=sum+sumDig
              (n div 10);
  sumDig:= sum
end;
```

Пример 3. Алгоритм Евклида, один из древнейших известных алгоритмов, предназначен для поиска наибольшего общего делителя (НОД) двух натуральных чисел. Формулируется он так.

Алгоритм Евклида. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

Этот алгоритм может быть сформулирован в рекурсивном виде. Во-первых, в нём для перехода к следующему шагу используется равенство НОД(a, b) = НОД($a - b$, b) при $a \geq b$. Кроме того, задано условие останова: если одно из чисел равно нулю, то НОД совпадает со вторым числом. Поэтому можно написать такую рекурсивную функцию:

```
алг цел NOD(цел a, b)
нач
  если a=0 или b=0 то
    знач:=a+b
    выход
  все
  если a>b то
       знач:=NOD(a-b, b)
  иначе знач:=NOD(a, b-a)
  все
кон
```

```
function NOD(a, b: integer):
                        integer;
begin
  if (a=0) or (b=0) then begin
    NOD:=a+b;
    exit
  end;
  if a>b then
       NOD:=NOD(a-b, b)
  else NOD:=NOD(a, b-a)
end;
```

Заметим, что при равенстве одного из чисел нулю второе число совпадает с суммой двух чисел, поэтому в качестве результата функции принимается $a + b$.

Существует и более быстрый вариант алгоритма Евклида (модифицированный алгоритм), в котором большее число заменяется на остаток от деления большего числа на меньшее, пока этот остаток не станет равным нулю.

Как работает рекурсия

Рассмотрим вычисление *факториала* $N!$: так называют произведение всех натуральных чисел от 1 до заданного числа N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$. Факториал может быть также введён с помощью **рекуррентной формулы**, которая связывает факториал данного числа с факториалом предыдущего[1]:

$$N! = \begin{cases} 1, & \text{если } N = 1; \\ N \cdot (N-1)!, & \text{если } N \geq 2. \end{cases}$$

Здесь первая часть описывает базовый случай (условие окончания рекурсии), а вторая — переход к следующему шагу. Запи-

[1] В математике принято, что факториал нуля равен 1. Поэтому равенство 0! = 1 можно тоже принять за базовый случай.

шем соответствующую функцию на школьном алгоритмическом языке, добавив в начале и в конце операторы вывода:

```
алг цел Fact(цел N)                          -> Fact(3)
нач                                           -> Fact(2)
  вывод '-> Fact(', N, '); нс                  -> Fact(1)
  если N<=1 то                                 <- знач=1
    знач:=1                                   <- знач=2
  иначе знач:=N*Fact(N-1)                    <- знач=6
  все
  вывод '<- знач=', знач, нс
кон
```

Справа от программы показан протокол её работы при вызове `Fact(3)` (для наглядности сделаны отступы, показывающие вложенность вызовов). Из протокола видно, что вызов `Fact(2)` происходит раньше, чем заканчивается вызов `Fact(3)`. Это значит, что компьютеру нужно где-то (без помощи программиста) запомнить состояние программы (в том числе значения всех локальных переменных) и адрес, по которому нужно вернуться после завершения вызова `Fact(2)`. Для этой цели используется стек.

Стек (англ. *stack* — кипа, стопка) — особая область памяти, в которой хранятся локальные переменные и адреса возврата из процедур и функций.

Один из регистров процессора называется **указателем стека** (англ. *stack pointer*, **SP**) — в нём записан адрес последней занятой ячейки стека. При вызове процедуры в стек помещаются значения всех её параметров, адрес возврата и выделяется место под локальные переменные.

На рисунке 8.4, *а* показано начальное состояние стека, серым цветом выделены занятые ячейки. Когда функция `Fact` вызывается из основной программы с параметром 3, в стек записывается значение параметра, затем — адрес возврата A, и выделяется место для локальной переменной `знач`, в которую будет записан результат[1] (рис. 8.4, *б*). При втором и третьем вложенных вызовах в стек добавляются аналогичные блоки данных (рис. 8.4, *в* и *г*). Заметьте, что в нашем случае адрес возврата A_F (точка внутри функции *Fact* после рекурсивного вызова) в последних двух блоках будет один и тот же.

[1] Результат функции, как правило, передаётся в вызывающую программу через регистры, но во время работы функции хранится в стеке.

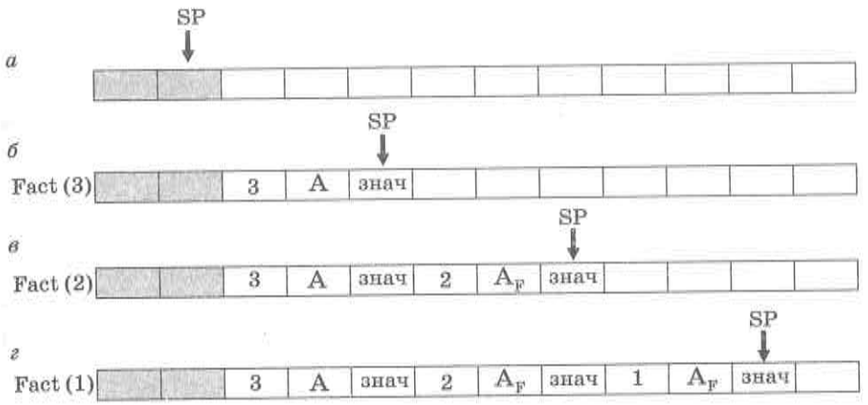


Рис. 8.4

Когда выполняется возврат из процедуры, состояние стека изменяется в обратную сторону: *г* — *в* — *б* — *а*.

Что же следует из этой схемы? Во-первых, с каждым новым вызовом расходуется дополнительная стековая память. Если вложенных вызовов будет очень много (или если процедура создаёт много локальных переменных), эта память закончится, и программа завершится аварийно.

Во-вторых, при каждом вызове процедуры некоторое время затрачивается на выполнение служебных операций (занесение данных в стек и т. п.), поэтому, как правило, рекурсивные программы выполняются несколько дольше, чем аналогичные нерекурсивные.

Всегда ли можно написать нерекурсивную программу? Оказывается, всегда. *Доказано*, что любой рекурсивный алгоритм может быть записан без использования рекурсии (хотя часто при этом программа усложняется и становится менее понятной).

Например, для вычисления факториала можно использовать обычный цикл:

```
знач:=1
нц для i от 2 до N
  знач:=знач*i
кц
```

В данном случае такой **итерационный** (т. е. повторяющийся, циклический) **алгоритм** значительно лучше, чем рекурсивный: он

не расходует стековую память и выполняется быстрее. Поэтому здесь нет никакой необходимости использовать рекурсию.

Итак, рекурсия — это мощный инструмент, заменяющий циклы в задачах, которые можно свести к более простым задачам того же типа. В сложных случаях использование рекурсии позволяет значительно упростить программу, сократить её текст и сделать более понятной.

Вместе с тем, если существует простое решение задачи без использования рекурсии, лучше применить именно его. Нужно стараться обходиться без рекурсии, если вложенность вызовов получается очень большой или подпрограмма использует много локальных данных.

Вопросы и задания

1. Что такое рекурсия? Приведите примеры.
2. Почему любое рекурсивное определение состоит из двух частей?
3. Что такое рекурсивная процедура (функция)?
4. Расскажите о задаче «Ханойские башни». Попытайтесь придумать алгоритм её решения, не использующий рекурсию.
5. Процедура *А* вызывает процедуру *Б*, а процедура *Б* — процедуру *А* и саму себя. Какую из этих процедур можно назвать рекурсивной?
6. В каком случае рекурсия никогда не остановится? Докажите, что в рассмотренных в параграфе задачах этого не случится.
7. Что такое стек? Как он используется при выполнении программ?
8. Почему при использовании рекурсии может случиться переполнение стека?
9. Назовите достоинства и недостатки рекурсии. Когда её следует использовать, а когда — нет?

Подготовьте сообщение

а) «Фракталы»
б) «Числа Фибоначчи»
в) «Рекурсия вокруг нас»
г) «Рекурсия в программировании: за и против»
д) «Рекурсия в произведениях искусства»

Задачи

1. Придумайте свою рекурсивную фигуру и опишите её.
*2. Используя графические возможности языка программирования, который вы изучаете, постройте на экране треугольник Серпинского и другие фракталы.
3. Напишите рекурсивную процедуру для перевода числа в двоичную систему, которая правильно работала бы для нуля (выводила бы 0).
*4. Дано натуральное число *N*. Требуется получить и вывести на экран все возможные *различные* способы представления этого числа в виде суммы натуральных чисел (т. е. 1 + 2 и 2 + 1 — это один и тот же способ разложения числа 3). Решите задачу с помощью рекурсивной процедуры.
5. Напишите рекурсивную процедуру для перевода числа из двоичной системы счисления в десятичную.
6. Напишите рекурсивную и нерекурсивную функции, вычисляющие НОД двух натуральных чисел с помощью модифицированного алгоритма Евклида. Какой вариант вы предпочтёте?

§ 62
Массивы

Что такое массив?

Основное предназначение современных компьютеров — обработка большого количества данных. При этом надо как-то обращаться к каждой из тысяч (или даже миллионов) ячеек с данными. Очень сложно дать каждой ячейке собственное имя и при этом не запутаться. Из этой ситуации выходят так: дают имя не ячейке, а группе ячеек, в которой каждая ячейка имеет собственный номер. Такая область памяти называется массивом.

Массив — это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер.

Массив — это сложный (составной) тип данных, он состоит из отдельных элементов, причём все они должны быть одного типа. Во многих языках программирования можно использовать «массивы массивов», т. е. массивы, элементы которых сами представляют собой массивы.

Для работы с массивами нужно в первую очередь научиться:

- выделять память нужного размера под массив;
- записывать данные в ячейку массива;
- читать данные из ячейки массива.

Чтобы использовать массив, надо его объявить: определить тип массива (тип входящих в него элементов), выделить место в

памяти и присвоить имя. Имена массивов строятся по тем же правилам, что и имена переменных.

В школьном алгоритмическом языке массивы называются **таблицами**. При их объявлении к названию типа данных добавляется слово **таб**:

```
целтаб A[1:5]
вещтаб V[0:5]
логтаб L[-5:5]
симтаб S[65:90]
```

В квадратных скобках через двоеточие записываются границы **индексов** — номеров ячеек массива. В приведённом примере массив A — это массив целых значений, ячейки имеют номера от 1 до 5. Массив вещественных значений V содержит 6 элементов с номерами от нуля[1] до 5. В логическом массиве L ячейки нумеруются от -5 до 5, а в символьном массиве S — от 65 до 90. В школьном алгоритмическом языке объявлять массивы (как и переменные) можно в любом месте программы.

В языке Паскаль массивы объявляются в блоке объявления переменных (выше ключевого слова **begin**). Объявление начинается ключевым словом **array** (в переводе с англ. — массив), после которого в квадратных скобках записывают минимальный и максимальный индексы, разделяя их двумя точками:

```
var A: array[1..5] of integer;
    V: array[0..5] of real;
    L: array[-5..5] of boolean;
    S: array[65..90] of char;
```

Индексы в Паскале могут быть не только целыми числами, но любыми значениями порядкового типа. Это значит, что для каждого значения можно указать предыдущее и следующее. Часто бывает удобно использовать символьные индексы. Например, в задаче подсчёта количества разных символов в файле можно использовать такой массив:

```
var Q: array['a'..'z'] of integer;
```

в котором индексы — это строчные латинские буквы.

Для того чтобы *обратиться к элементу массива*, нужно записать имя массива и в квадратных скобках — индекс нужного

[1] Нумерация с нуля часто используется в языках программирования, например в языке Си и родственных ему языках.

элемента, например $A[3]$. Индексом может быть также значение целой переменной или арифметического выражения, результат которого — целое число.

В следующем примере массив заполняется квадратами первых натуральных чисел:

```
алг Массив
нач
  цел i, N=10
  целтаб A[1:N]
  нц для i от 1 до N
    A[i]:=i*i
  кц
кон
```

```
program qq;
const N=10;
var A: array[1..N] of integer;
    i: integer;
begin
  for i:=1 to N do
    A[i]:=i*i;
end.
```

В школьном алгоритмическом языке можно при объявлении массива указывать вычисляемые границы индексов, зависящие от переменных. В данном случае массив A состоит из 10 элементов, потому что к моменту его объявления в переменной N было число 10.

В Паскале при объявлении границ индексов массивов можно использовать **константы** — постоянные величины, имеющие имя. В приведённом примере с помощью ключевого слова **const** объявлена константа N, равная 10. Константы обычно вводятся выше блока объявления переменных. Использование констант очень удобно, потому что при изменении размера массива в программе нужно поменять только одно число — значение этой константы.

Далее во всех примерах мы будем считать, что в программе объявлен целочисленный массив A с индексами от 1 до N, а также целочисленная переменная i, которая будет обозначать индекс элемента массива. Чтобы ввести такой массив или вывести его на экран, нужно использовать цикл, т. е. ввод и вывод массива выполняется поэлементно:

```
нц для i от 1 до N
  вывод 'A[',i,']='
  ввод A[i]
кц
...
нц для i от 1 до N
  вывод A[i], ' '
кц
```

```
for i:=1 to N do begin
  write('A[',i,']=');
  read(A[i])
end;
...
for i:=1 to N do
  write(A[i],' ');
```

В этом примере перед вводом очередного элемента массива на экран выводится подсказка. Например, при вводе 3-го элемента будет выведено: A[3]=. После вывода каждого элемента ставится пробел, иначе все значения сольются в одну строку.

В учебных примерах массивы часто заполняют случайными числами:

```
нц для i от 1 до N
  A[i]:=irand(20,100)
  вывод A[i], ' '
кц
```

```
for i:=1 to N do begin
  A[i]:=20+random(81);
  write(A[i], ' ')
end;
```

Перебор элементов

Перебор элементов состоит в том, что мы в цикле просматриваем все элементы массива и, если нужно, выполняем с каждым из них некоторую операцию. Для этого удобнее всего использовать цикл с переменной, которая изменяется от минимального до максимального индекса. Для массива, элементы которого имеют индексы от 1 до N, цикл выглядит так:

```
нц для i от 1 до N
 ...
кц
```

```
for i:=1 to N do begin
 ...
end;
```

Здесь вместо многоточия можно добавлять операторы, работающие с элементом $A[i]$.

Во многих задачах нужно найти в массиве все элементы, удовлетворяющие заданному условию, и как-то их обработать. Простейшая из таких задач — подсчёт нужных элементов. Для решения этой задачи нужно ввести переменную-счётчик, начальное значение которой равно нулю. Далее в цикле (от 1 до N) просматриваем все элементы массива. Если для очередного элемента выполняется заданное условие, то увеличиваем счётчик на 1. На псевдокоде этот алгоритм выглядит так:

```
счётчик:=0
нц для i от 1 до N
  если условие выполняется для A[i] то
    счётчик:=счётчик+1
  все
кц
```

Предположим, что в массиве A записаны данные о росте игроков баскетбольной команды (в сантиметрах). Найдем количество

игроков, рост которых больше 180 см, но меньше 190 см. В следующей программе используется переменная-счётчик *count*:

```
цел count=0
нц для i от 1 до N
  если 180<A[i] и A[i]<190 то
    count:=count+1
  все
кц
```

```
count:=0;
for i:=1 to N do
  if (180<A[i])
     and (A[i]<190)
  then count:=count+1;
```

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

```
цел count=0, sum=0
нц для i от 1 до N
  если 180<A[i] и A[i]<190 то
    count:=count+1
    sum:=sum+A[i]
  все
кц
вывод sum/count
```

```
count:=0; sum:=0;
for i:=1 to N do
  if (180<A[i])
     and (A[i]<190)
  then begin
    count:=count+1;
    sum:=sum+A[i]
  end;
write(sum/count);
```

Вопросы и задания

1. Что такое массив? Зачем нужны массивы?
2. Зачем нужно объявлять массивы?
3. Как объявляются массивы в школьном алгоритмическом языке и в Паскале?
4. Как вы думаете, почему элементы массива расположены в памяти рядом?
5. Как обращаются к элементу массива?
6. Могут ли индексы элементов массива начинаться с 0? С –5?
7. Почему размер массива лучше вводить как константу, а не как число?
8. Как ввести массив и вывести его на экран?
9. Как заполнить массив случайными числами в диапазоне от 100 до 200?

Подготовьте сообщение

а) «Массивы в языке Си»

б) «Списки и словари в языке Python»

Задачи

1. Заполните массив элементами арифметической прогрессии. Её первый элемент и разность нужно ввести с клавиатуры.
2. Заполните массив степенями числа 2 (от 2^1 до 2^N).
3. Заполните массив первыми числами Фибоначчи.
*4. Заполните массив из N элементов случайными целыми числами в диапазоне 1..N так, чтобы в массив обязательно вошли все числа от 1 до N (постройте случайную перестановку).
*5. Постройте случайную перестановку чисел от 1 до N так, чтобы первое число обязательно было равно 5 ($N \geq 5$).
6. Заполните массив случайными числами в диапазоне 20..100 и подсчитайте отдельно число элементов с чётными и нечётными значениями.
7. Заполните массив случайными числами в диапазоне 1000..2000 и подсчитайте число элементов, у которых вторая с конца цифра — чётная.
8. Заполните массив случайными числами в диапазоне 0..100 и подсчитайте отдельно среднее значение всех элементов, меньших 50, и среднее значение всех элементов, которые больше или равны 50.

§ 63 Алгоритмы обработки массивов

Поиск в массиве

Требуется найти в массиве элемент, значение которого равно значению переменной X, или сообщить, что такого элемента в массиве нет. Алгоритм решения сводится к просмотру всех элементов массива с первого до последнего. Как только будет найден элемент, равный X, нужно выйти из цикла и вывести результат. Напрашивается такой алгоритм:

```
i:=1
нц пока A[i]<>X
  i:=i+1
кц
вывод 'A[', i, ']=', X
```

Он хорошо работает, если нужный элемент в массиве есть, однако приведёт к ошибке, если такого элемента нет, — получится зацикливание и выход за границы массива. Поэтому в условие нужно добавить ещё одно ограничение: i<=N. Если после окончания цикла это условие нарушено, значит, поиск был неудачным — элемента нет:

```
i:=1
нц пока i<=N и A[i]<>X
  i:=i+1
кц
если i<=N то
    вывод 'A[',i,']=',X
иначе вывод 'Не нашли!'
все
```

```
i:=1;
while (i<=N) and (A[i]<>X) do
  i:=i+1;
if i<=N then
    write('A[', i, ']=', X)
else write('Не нашли!');
```

Отметим одну тонкость. В сложном условии i<=N и A[i]<>X первым должно проверяться именно отношение i<=N. Если первая часть условия, образованного с помощью операции «И», ложно, то вторая часть, как правило[1], не вычисляется — уже понятно, что всё условие ложно. Дело в том, что если i>N, проверка условия A[i]<>X приводит к *выходу за границы массива*, и программа может завершиться аварийно.

Возможен ещё один поход к решению этой задачи: используя цикл с переменной, перебирать все элементы массива и досрочно завершить цикл, если найдено требуемое значение.

```
nX:=0
нц для i от 1 до N
  если A[i]=X то
    nX:=i
    выход
  все
кц
если nX>0 то
    вывод 'A[', nX, ']=', X
иначе вывод 'Не нашли!'
все
```

```
nX:=0;
for i:=1 to N do
  if A[i]=X then begin
    nX:=i;
    break
  end;
if nX>0 then
    write('A[', nX, ']=', X)
else write('Не нашли!');
```

Здесь для выхода из цикла используется оператор выход (в Паскале — break), номер найденного элемента сохраняется в переменной nX. Если её значение осталось равным нулю (не изменилось в ходе выполнения цикла), то в массиве нет элемента, равного X.

[1] Это зависит от транслятора и его настроек. В школьном алгоритмическом языке и во многих других современных языках (например, в C, C++, Python, Javascript, PHP) такое поведение гарантировано стандартом, а в Паскале — нет.

Максимальный элемент

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

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

```
M:=A[1]
нц для i от 2 до N
  если A[i]>M то
    M:=A[i]
  все
кц
вывод M
```

```
M:=A[1];
for i:=2 to N do
  if A[i]>M then
    M:=A[i];
write(M);
```

Теперь предположим, что нужно найти не только значение, но и номер максимального элемента. Казалось бы, нужно ввести ещё одну переменную $nMax$ для хранения номера, сначала записать в неё 1 (считаем первый элемент максимальным) и затем, когда найдём новый максимальный элемент, запоминать его номер в переменной $nMax$:

```
M:=A[i]; nMax:=1
нц для i от 2 до N
  если A[i]>M то
    M:=A[i]
    nMax:=i
  все
кц
вывод 'A[', nMax, ']=', M
```

```
M:=A[1]; nMax:=1;
for i:=2 to N do
  if A[i]>M then begin
    M:=A[i];
    nMax:=i
  end;
write('A[', nMax, ']=' ,M);
```

Однако это не самый лучший вариант. Дело в том, что по номеру элемента можно всегда определить его значение. Поэтому достаточно хранить только номер максимального элемента[1]. Если

[1] Однако нужно учитывать, что обращение к элементу массива выполняется медленнее, чем обращение к переменной.

этот номер равен $nMax$, то значение максимального элемента равно $A[nMax]$:

```
nMax:=1
нц для i от 2 до N
  если A[i]>A[nMax] то
    nMax:=i
  все
кц
вывод 'A[',nMax,']=',A[nMax]
```

```
nMax:=1;
for i:=2 to N do
  if A[i]>A[nMax] then
    nMax:=i;
write('A[', nMax, ']=',
                A[nMax]);
```

Реверс массива

Реверс массива — это перестановка его элементов в обратном порядке: первый элемент становится последним, а последний — первым (рис. 8.5).

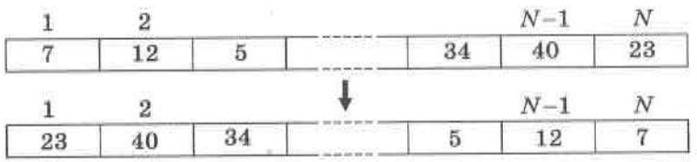


Рис. 8.5

Из рисунка 8.5 следует, что 1-й элемент меняется местами с N-м, второй — с $(N - 1)$-м и т. д. Сумма индексов элементов, участвующих в обмене, для всех пар равна $N + 1$, поэтому элемент с номером i должен меняться местами с $(N + 1 - i)$-м элементом. Кажется, что можно написать такой цикл:

```
нц для i от 1 до N
  поменять местами A[i] и A[N+1-i]
кц
```

Однако это неверно. Посмотрим, что получится для массива из четырёх элементов (рис. 8.6).

Как видите, массив вернулся в исходное состояние: реверс выполнен дважды. Поэтому нужно остановить цикл на середине массива:

```
нц для i от 1 до div(N,2)
  поменять местами A[i] и A[N+1-i]
кц
```

	1	2	3	4
	7	12	40	23
A[1]↔A[4]	23	12	40	7
A[2]↔A[3]	23	40	12	7
A[3]↔A[2]	23	12	40	7
A[4]↔A[1]	7	12	40	23

Рис. 8.6

Для обмена используется вспомогательная целая переменная c:

```
нц для i от 1 до div(N,2)
  c:=A[i]
  A[i]:=A[N+1-i]
  A[N+1-i]:=c
кц
```

```
for i:=1 to N div 2 do begin
  c:=A[i];
  A[i]:=A[N+1-i];
  A[N+1-i]:=c
end;
```

Сдвиг элементов массива

При удалении и вставке элементов необходимо выполнять **сдвиг** части или всех элементов массива в ту или другую сторону. Массив часто рисуют в виде таблицы, где первый элемент расположен слева. Поэтому сдвиг влево — это перемещение всех элементов на одну ячейку, при котором $A[2]$ переходит на место $A[1]$, $A[3]$ — на место $A[2]$ и т. д. (рис. 8.7).

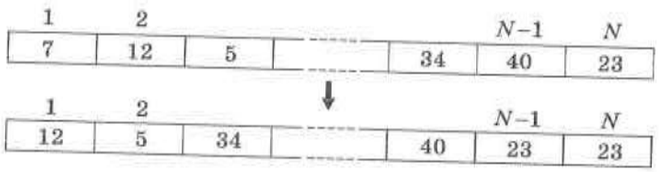


Рис. 8.7

Последний элемент остаётся на своём месте, поскольку новое значение для него взять неоткуда — массив кончился. Алгоритм выглядит так:

```
нц для i от 1 до N-1
  A[i]:=A[i+1]
кц
```

```
for i:=1 to N-1 do
  A[i]:=A[i+1];
```


Обратите внимание, что цикл заканчивается при $i = N - 1$ (а не N), чтобы не было выхода за границы массива, т. е. обращения к несуществующему элементу $A[N+1]$.

При таком сдвиге первый элемент пропадает, а последний дублируется. Можно старое значение первого элемента записать на место последнего. Такой сдвиг называется **циклическим** (см. § 28). Предварительно (до начала цикла) первый элемент нужно запомнить во вспомогательной переменной, а после завершения цикла записать его в последнюю ячейку массива:

```
c:=A[1]
нц для i от 1 до N-1
  A[i]:=A[i+1]
кц
A[N]:=c
```

```
c:=A[1];
for i:=1 to N-1 do
  A[i]:=A[i+1];
A[N]:=c;
```

Отбор нужных элементов

Требуется отобрать все элементы массива A, удовлетворяющие некоторому условию, в массив B. «Очевидное» решение:

```
нц для i от 1 до N
  если условие выполняется для A[i] то
    B[i]:=A[i]
  все
кц
```

На самом деле это решение неудачное, потому что нужные элементы в массиве B оказываются расположенными вразброс, на тех местах, где они стояли в массиве A. Поэтому работать с таким массивом B очень неудобно. На рисунке 8.8 изображён случай, когда отбираются элементы с чётными значениями.

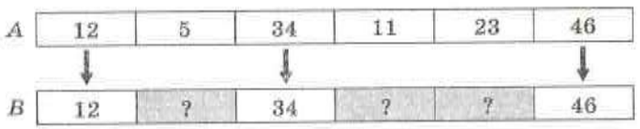


Рис. 8.8

Будет удобно, если все отобранные элементы будут стоять в начале массива B (рис. 8.9).

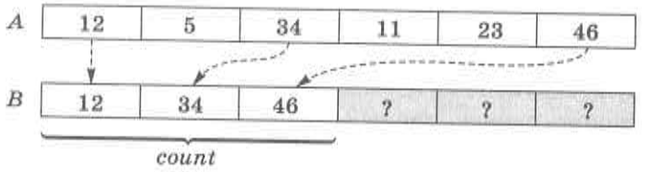


Рис. 8.9

Для этого вводят переменную-счётчик, в которой считают количество найденных элементов. Сначала её значение равно нулю. Как только очередной подходящий элемент найден, счётчик увеличивается на 1, и теперь значение счётчика — это номер первой свободной ячейки массива *B*, в неё и записывается найденное значение. Программа, отбирающая все элементы с чётными значениями, выглядит так:

```
count:=0
нц для i от 1 до N
  если mod(A[i],2)=0 то
    count:=count+1
    B[count]:=A[i]
  все
кц
```

```
count:=0;
for i:=1 to N do
  if A[i] mod 2=0 then begin
    count:=count+1;
    B[count]:=A[i]
  end;
```

Нужно помнить, что только первые *count* элементов массива *B* рабочие, остальные содержат неизвестные данные. Например, вот так можно вывести найденные элементы на экран:

```
нц для i от 1 до count
  вывод B[i], ' '
кц
```

```
for i:=1 to count do
  write(B[i], ' ');
```

Если вместо массива *B* использовать тот же массив *A*, где находятся исходные числа, все «нужные» элементы будут сгруппированы в начале, а их количество записано в переменной *count* (рис. 8.10).

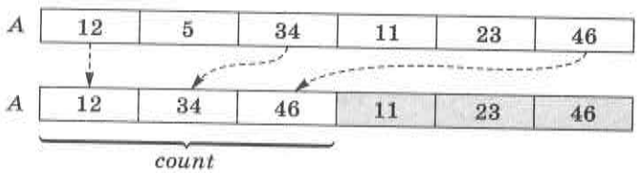


Рис. 8.10

Вопросы и задания

1. Почему при поиске индекса максимального элемента необязательно хранить само значение максимального элемента?
2. Что такое реверс массива?
3. Как вы думаете, какую ошибку чаще всего делают начинающие программисты, программируя реверс массива?
4. Как вы думаете, какие проблемы (и ошибки) могут возникнуть при циклическом сдвиге массива *вправо*?
5. Что произойдёт с массивом при выполнении следующего фрагмента программы?

```
нц для i от 1 до N-1
  A[i+1]:=A[i]
кц
```

```
for i:=1 to N-1 do
  A[i+1]:=A[i];
```

6. Как при использовании приведённых алгоритмов поиска определить, что элемент не найден?
7. Что такое выход за границы массива? Почему он может быть опасен?
8. Опишите «очевидный» алгоритм отбора части элементов одного массива в другой массив. Почему его не используют?

Подготовьте сообщение

а) «Выход за границы массива»
б) «Алгоритмы работы со списками на языке Python»

Задачи

1. Напишите программу, которая находит максимальный и минимальный из элементов массива с чётными положительными значениями. Если в массиве нет элементов с чётными положительными значениями, нужно вывести сообщение об этом.
2. Введите массив с клавиатуры и найдите (за один проход) количество элементов, имеющих максимальное значение.
3. Найдите за один проход по массиву три его различных элемента, которые меньше всех остальных («три минимума»).

*4. Заполните массив случайными числами в диапазоне 10..12 и найдите длину самой длинной последовательности стоящих рядом элементов с одинаковыми значениями.

5. Заполните массив случайными числами в диапазоне 0..4 и выведите на экран номера всех элементов, значение которых равно *X* (*X* вводится с клавиатуры).

6. Заполните массив случайными числами и переставьте соседние элементы, поменяв 1-й элемент со 2-м, 3-й — с 4-м и т. д.
7. Заполните массив с чётным количеством элементов случайными числами и выполните реверс отдельно для первой и второй половин массива.
8. Заполните массив случайными числами и выполните реверс для части массива между элементами с индексами K и M (включая эти элементы).
9. Напишите программу для выполнения циклического сдвига массива вправо на K элементов.
10. Напишите программу, которая заполняет массив случайными целыми числами, вводит с клавиатуры целое число N и копирует в новый массив все числа из исходного массива, для которых сумма значений всех цифр равна N.
11. Найдите в массиве все простые числа и скопируйте их в новый массив.

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

§ 64 Сортировка

Сортировка — это расстановка элементов массива в заданном порядке.

Порядок сортировки может быть любым, для чисел обычно рассматривают сортировку по возрастанию (или убыванию) значений.

Возникает естественный вопрос: зачем сортировать данные? На него легко ответить, вспомнив, например, работу со словарями: сортировка слов по алфавиту облегчает поиск нужной информации.

Программисты изобрели множество способов сортировки. В целом их можно разделить на две группы: 1) простые, но медленно работающие (на больших массивах) и 2) сложные, но быстрые. Мы изучим два классических метода из первой группы и один метод из второй — знаменитую «быструю сортировку», предложенную *Ч. Хоаром*.

Далее мы будем рассматривать сортировку массива по возрастанию (или убыванию) значений. Для массивов, в которых есть одинаковые элементы, используются понятия «сортировка по не-

убыванию» (каждый следующий элемент не меньше, чем предыдущий) и «сортировка по невозрастанию».

Метод пузырька (сортировка обменами)

Название этого метода произошло от известного физического явления — пузырёк воздуха в воде поднимается вверх. Если говорить о сортировке массива, сначала поднимается «наверх» (к началу массива) самый «лёгкий» элемент (элемент с минимальными значениями), затем следующий и т. д.

Сначала сравниваем последний элемент с предпоследним. Если они стоят неправильно (меньший элемент «ниже»), то меняем их местами. Далее так же рассматриваем следующую пару элементов и т. д. (рис. 8.11).

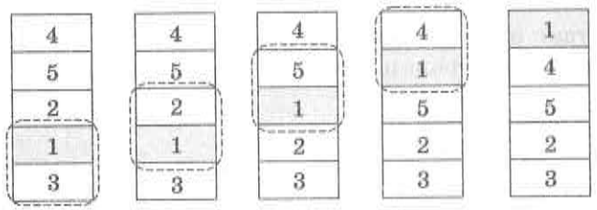


Рис. 8.11

После обработки пары ($A[1]$, $A[2]$) минимальный элемент стоит на месте $A[1]$. Это значит, что на следующих этапах его можно не рассматривать. Первый цикл, устанавливающий на свое место первый (минимальный) элемент, можно на псевдокоде записать так:

```
нц для j от N-1 до 1 шаг -1
  если A[j+1]<A[j] то
    поменять местами A[j] и A[j+1]
  все
кц
```

Здесь j — целочисленная переменная. Обратите внимание, что на очередном шаге сравниваются элементы $A[j]$ и $A[j+1]$, поэтому цикл начинается с $j = N - 1$. Если начать с $j = N$, то на первом же шаге получаем выход за границы массива — обращение к элементу $A[N+1]$.

За один проход такой цикл ставит на место один элемент. Чтобы «подтянуть» второй элемент, нужно написать ещё один почти такой же цикл, который будет отличаться только конечным

значением *j* в заголовке цикла. Так как верхний элемент уже стоит на месте, его не нужно трогать:

```
нц для j от N-1 до 2 шаг -1
  если A[j+1]<A[j] то
    поменять местами A[j] и A[j+1]
  все
кц
```

При установке 3-го элемента конечное значение для *j* будет 3 и т. д.

Таких циклов нужно сделать $N - 1$: на 1 меньше, чем количество элементов массива. Почему не N? Дело в том, что если $N - 1$ элементов поставлены на свои места, то оставшийся автоматически встает на своё место — другого места нет. Поэтому полный алгоритм сортировки представляет собой такой вложенный цикл:

```
нц для i от 1 до N-1
  нц для j от N-1 до i шаг -1
    если A[j+1]<A[j] то
      поменять местами A[j] и A[j+1]
    все
  кц
кц
```

Записать полную программу на выбранном языке вы можете самостоятельно.

Метод выбора

Ещё один популярный простой метод сортировки — метод выбора, при котором на каждом этапе выбирается минимальный элемент (из оставшихся) и ставится на свое место. Алгоритм в общем виде можно записать так:

```
нц для i от 1 до N-1
  найти номер nMin минимального элемента из A[i]..A[N]
  если i<>nMin то
    поменять местами A[i] и A[nMin]
  все
кц
```


Здесь перестановка происходит только тогда, когда найденный минимальный элемент стоит не на своём месте, т. е. i<>nMin. Поскольку поиск минимального элемента выполняется в цикле, этот алгоритм сортировки также представляет собой вложенный цикл:

```
нц для i от 1 до N-1
  nMin:= i
  нц для j от i+1 до N
    если A[j]<A[nMin] то
      nMin:=j
    все
  кц
  если i<>nMin то
    поменять местами A[i] и A[nMin]
  все
кц
```

«Быстрая сортировка»

Один из самых популярных «быстрых» алгоритмов, разработанный в 1960 г. английским учёным *Чарльзом Хоаром*, так и называется — «быстрая сортировка» (англ. *quicksort*).

Ч. Э. Хоар (род. в 1934)

Будем исходить из того, что сначала лучше делать перестановки элементов массива на большом расстоянии. Предположим, что у нас есть N элементов и известно, что они уже отсортированы в обратном порядке. Тогда за $N/2$ обменов можно отсортировать их как нужно — сначала поменять местами первый и последний, а затем последовательно двигаться с двух сторон к центру. Хотя это справедливо только тогда, когда порядок элементов обратный, подобная идея положена в основу алгоритма Quicksort.

Пусть дан массив A из N элементов. Выберем сначала наугад любой элемент массива (назовем его X). На первом этапе мы рас-

ставим элементы так, что слева от некоторой границы находятся все числа, меньшие или равные X, а справа — большие или равные X. Заметим, что элементы, равные X, могут находиться в обеих частях.

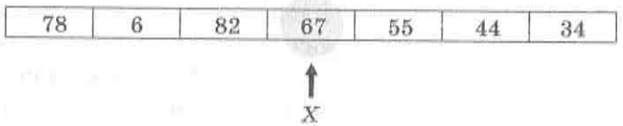
$A[i] \le X$	$A[i] \ge X$

Теперь элементы расположены так, что ни один элемент из первой части при сортировке не окажется во второй части и наоборот. Поэтому далее достаточно отсортировать отдельно каждую часть массива. Такой подход называют «разделяй и властвуй» (англ. *divide and conquer*).

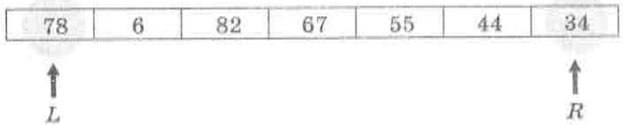
Лучше всего выбирать X так, чтобы в обеих частях было равное количество элементов. Такое значение X называется *медианой массива*. Однако для того, чтобы найти медиану, надо сначала отсортировать массив, т. е. заранее решить ту самую задачу, которую мы собираемся решить этим способом. Поэтому обычно в качестве X выбирают средний элемент массива (или элемент со случайным номером).

Сначала будем просматривать массив слева до тех пор, пока не обнаружим элемент, который больше X (и, следовательно, должен стоять справа от X). Затем просматриваем массив справа до тех пор, пока не обнаружим элемент, меньший X (он должен стоять слева от X). Теперь поменяем местами эти два элемента и продолжим просмотр до тех пор, пока два «просмотра» не встретятся где-то в середине массива. В результате массив окажется разбитым на 2 части: левую со значениями, меньшими или равными X, и правую со значениями, большими или равными X. На этом первый этап («разделение») закончен. Затем такая же процедура применяется к обеим частям массива до тех пор, пока в каждой части не останется по одному элементу (таким образом, массив будет отсортирован).

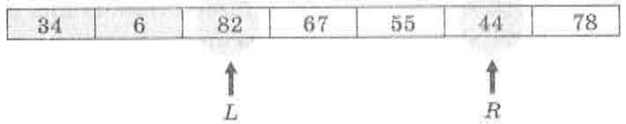
Чтобы понять сущность метода, рассмотрим пример. Пусть задан массив:



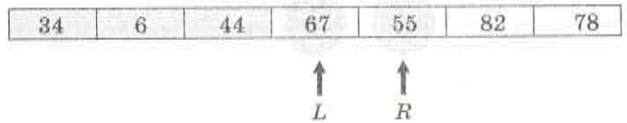
Выберем в качестве X средний элемент массива, т. е. 67. Найдем первый слева элемент массива, который больше или равен X и должен стоять во второй части. Это число 78. Обозначим индекс этого элемента через L. Теперь находим самый правый элемент, который меньше X и должен стоять в первой части. Это число 34. Обозначим его индекс через R:



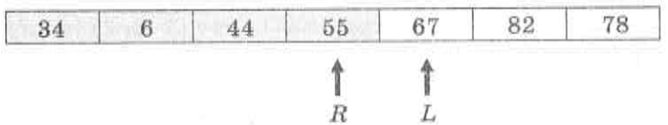
Теперь поменяем местами эти два элемента. Сдвигая переменную L вправо, а R — влево, находим следующую пару, которую надо переставить. Это числа 82 и 44:



Следующая пара элементов для перестановки — числа 67 и 55:



После этой перестановки дальнейший поиск приводит к тому, что переменная L становится больше R:



В результате все элементы массива, расположенные левее $A[L]$, меньше или равны X, а все элементы правее $A[R]$ — больше или равны X.

Теперь нужно применить тот же алгоритм к двум полученным частям массива: первая часть — с 1-го элемента до R-го элемента, вторая часть — с L-го до последнего элемента. Как вы знаете, такой приём называется *рекурсией*.

Чтобы не загромождать решение деталями оформления, которые сильно различаются в школьном алгоритмическом языке и в Паскале, предположим, что в нашей программе один **глобальный** массив A с индексами от 1 до N, который нужно сортировать. Термин «глобальный» означает, что массив доступен всем процедурам и функциям. Глобальные данные объявляются выше основной программы:

```
цел N = 5
целтаб A[1:N]
алг Быстрая сортировка
нач
 ...
кон
```

Тогда процедура сортировки принимает только два аргумента, ограничивающие её «рабочую зону»: *nStart* — номер первого элемента, и *nEnd* — номер последнего элемента. Если *nStart* = *nEnd*, то в «рабочей зоне» один элемент и сортировка не требуется, т. е. нужно выйти из процедуры. В этом случае рекурсивные вызовы заканчиваются.

Приведём полностью процедуру быстрой сортировки на школьном алгоритмическом языке:

```
алг qSort(цел nStart, nEnd)
нач
  цел L, R, c, X
  если nStart>=nEnd то выход все
  L:=nStart; R:=nEnd
  X:=A[div(L+R,2)] | или :=A[L+div(R-L,2)]
  нц пока L<=R | разделение
    нц пока A[L]<X; L:=L+1 кц
    нц пока A[R]>X; R:=R-1 кц
    если L<=R то
      c:=A[L]; A[L]:=A[R]; A[R]:=c
      L:=L+1; R:=R-1
    все
  кц
  qSort(nStart, R) | рекурсивные вызовы
  qSort(L, nEnd)
кон
```


Для того чтобы отсортировать весь массив, нужно вызвать эту процедуру так:

```
qSort(1, N)
```

Скорость работы быстрой сортировки зависит от того, насколько удачно выбирается вспомогательный элемент X. Самый лучший случай — когда на каждом этапе массив делится на две равные части. Худший случай — когда в одной части оказывается только один элемент, а в другой — все остальные. При этом глубина рекурсии достигает N, что может привести к переполнению стека (нехватке стековой памяти).

Для того чтобы уменьшить вероятность худшего случая, в алгоритм вводят случайность: в качестве X на каждом шаге выбирают не середину рабочей части массива, а элемент со случайным номером:

```
X:=A[irand(L,R)]
```

В таблице 8.1 сравнивается время сортировки (в секундах) массивов разного размера, заполненных случайными значениями, с использованием трёх изученных алгоритмов.

Таблица 8.1

N	Метод пузырька	Метод выбора	Быстрая сортировка
1000	0,24 с	0,12 с	0,004 с
5000	5,3 с	2,9 с	0,024 с
15000	45 с	34 с	0,068 с

Как показывают эти данные, преимущество быстрой сортировки становится подавляющим при увеличении N.

Вопросы и задания

1. Что такое сортировка?
2. На какой идее основан метод пузырька? Метод выбора?
3. Объясните, зачем нужен вложенный цикл в описанных методах сортировки.
4. Сравните на примере метод пузырька и метод выбора. Какой из них требует меньше перестановок?

5. Расскажите про основные идеи метода «быстрой сортировки».
6. Как нужно изменить приведённые в параграфе алгоритмы, чтобы элементы массива были отсортированы по убыванию?
7. Как вы думаете, можно ли использовать метод «быстрой сортировки» для нечисловых данных, например для символьных строк?
8. От чего зависит скорость «быстрой сортировки»? Какой самый лучший и самый худший случай?
9. Как вы думаете, может ли метод «быстрой сортировки» работать дольше, чем метод выбора (или другой «простой» метод)? Если да, то при каких условиях?

Подготовьте сообщение

а) «Сортировка вставкой»
б) «Сортировка слиянием»
в) «Сортировка списков на языке Python»

Задачи

1. Отсортировать массив и найти количество различных чисел в нём.
2. Напишите программу, в которой сортировка выполняется «методом камня» — самый «тяжёлый» элемент опускается в конец массива.
3. Напишите программу, которая выполняет неполную сортировку массива: ставит в начало массива три самых меньших по величине элемента в порядке возрастания (неубывания).
4. Напишите вариант метода пузырька, который заканчивает работу, если на очередном шаге внешнего цикла не было перестановок.
5. Напишите программу, которая сортирует массив по возрастанию последней цифры числа.
6. Напишите программу, которая сортирует массив по убыванию суммы цифр числа.
7. Напишите рекурсивные варианты процедур сортировки методом пузырька и методом выбора.
8. Напишите программу, которая сортирует первую половину массива по возрастанию, а вторую — по убыванию (элементы из первой половины не должны попадать во вторую и наоборот).
9. Напишите программу, которая сортирует массив, а затем находит максимальное из чисел, встречающихся в массиве несколько раз.
*10. Напишите программу, которая сравнивает количество перестановок при сортировке одного и того же массива разными методами. Проведите эксперименты для возрастающей последовательности (уже отсортированной), убывающей (отсортированной в обратном порядке) и случайной.

§ 65 Двоичный поиск

Ранее мы уже рассматривали задачу поиска элемента в массиве и привели алгоритм, который сводится к просмотру всех элементов массива. Такой поиск называют **линейным**. Для массива из 1000 элементов нужно сделать 1000 сравнений, чтобы убедиться, что заданного элемента в массиве нет. Если число элементов (например, записей в базе данных) очень велико, время поиска может оказаться недопустимым, потому что пользователь не дождётся ответа.

Как вы помните, основная задача сортировки — облегчить последующий поиск данных. Вспомним, как мы ищем нужное слово (например, слово «гравицапа») в словаре. Сначала открываем словарь примерно в середине и смотрим, какие там слова. Если они начинаются на букву «Л», то слово «гравицапа» явно находится на предыдущих страницах, и вторую часть словаря можно не смотреть. Теперь так же проверяем страницу в середине первой половины, и т. д. Такой поиск называется **двоичным**. Понятно, что он возможен только тогда, когда данные предварительно отсортированы. Для примера на рис. 8.12 показан поиск числа $X = 44$ в отсортированном массиве.

Серым фоном выделены ячейки, которые уже не рассматриваются, потому что в них не может быть заданного числа. Перемен-

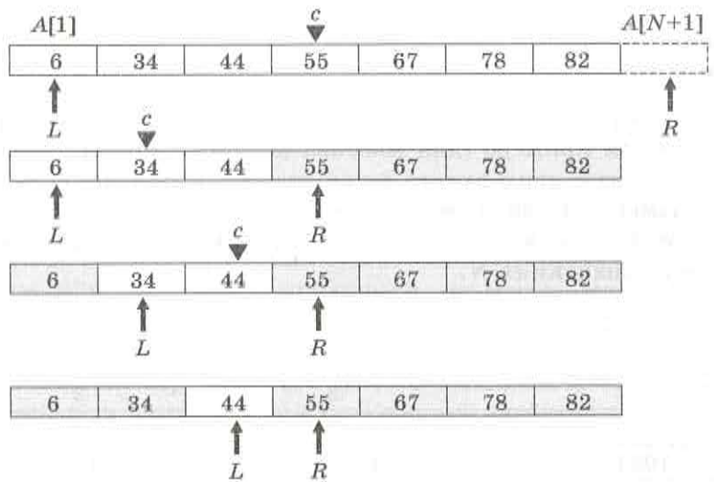


Рис. 8.12

ные L и R ограничивают «рабочую зону» массива: L содержит номер первого элемента, а R — номер элемента, *следующего за последним*. Таким образом, нужный элемент (если он есть) находится в части массива, которая начинается с элемента $A[L]$ и заканчивается элементом $A[R–1]$.

На каждом шаге вычисляется номер среднего элемента «рабочей зоны», он записывается в переменную c. Если $X<A[c]$, то этот элемент может находиться только левее $A[c]$, и правая граница R перемещается в c. В противном случае нужный элемент находится правее середины или совпадает с $A[c]$; при этом левая граница L перемещается в c.

Поиск заканчивается при выполнении условия $L = R - 1$, когда в рабочей зоне остаётся один элемент. Если при этом $A[L] = X$, то в результате найден элемент, равный X, иначе такого элемента нет.

```
цел L, R, c
L:=1; R:=N+1
нц пока L<R-1
  c:=div(L+R,2)
  если X<A[c] то
        R:=c
  иначе L:=c
  все
кц
если A[L]=X то
      вывод 'A[',L,']=',X
иначе вывод 'Не нашли!'
все
```

Двоичный поиск работает значительно быстрее, чем линейный. В нашем примере (для массива из 8 элементов) удаётся решить задачу за 3 шага вместо 8 при линейном поиске. При увеличении размера массива эта разница становится впечатляющей. В таблице 8.2 сравнивается количество шагов для двух методов при разных значениях N.

Таблица 8.2

N	Линейный поиск	Двоичный поиск
2	2	2
16	16	5
1024	1024	11
1048576	1048576	21

Однако при этом нельзя сказать, что двоичный поиск лучше линейного. Нужно помнить, что данные необходимо предварительно отсортировать, а это может занять значительно больше времени, чем сам поиск. Поэтому такой подход эффективен, если данные меняются (и сортируются) редко, а поиск выполняется часто. Такая ситуация характерна, например, для баз данных.

Вопросы и задания

1. Почему приведённый в параграфе алгоритм поиска называется двоичным?
2. Как можно примерно подсчитать количество шагов при двоичном поиске?

Подготовьте сообщение

а) «Двоичный поиск в нашей жизни»
б) «Линейный и двоичный поиск: достоинства и недостатки»

Задачи

1. Напишите программу, которая сортирует массив по убыванию и ищет в нём все значения, равные введённому числу.
2. Напишите программу, которая считает среднее число шагов при двоичном поиске для массива из 32 элементов в диапазоне 0..100. Для поиска используйте 1000 случайных чисел в этом же диапазоне.

§ 66 Символьные строки

Что такое символьная строка?

Если в середине XX века первые компьютеры использовались, главным образом, для выполнения сложных математических расчётов, сейчас их основная работа — обработка текстовой (символьной) информации.

Символьная строка — это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Для работы с символами во многих языках программирования есть переменные специального типа: символы и символьные массивы. Казалось бы, массив — это и есть символьная строка, однако в школьном алго-

ритмическом языке и в Паскале для строк используются специальные типы данных.

Почему возникла потребность в создании такого специального типа данных? Дело в том, что массив — это группа символов, каждый из которых независим от других. Это значит, что (в школьном алгоритмическом языке и в Паскале) вводить символьный массив нужно посимвольно, в цикле. Более того, размер массива задается при объявлении, и не очень ясно, как использовать массивы для работы со строками переменной длины. Поэтому нужен новый тип данных, который позволяет:

- работать с целой символьной строкой как с единым объектом;
- использовать строки переменной длины.

Строка, как и массив — это сложный тип данных, она состоит из отдельных символов.

Тип данных для работы с символьными строками в школьном алгоритмическом языке называется **литерным** и обозначается **лит** (от слова «литерный» — буквенный), а в Паскале — строковым и обозначается **string** (в переводе с англ. — строка). Вот пример объявления строки:

```
лит s
```

```
var s: string;
```

Для того чтобы записать в строку значение, используют оператор присваивания:

```
s:='Вася пошёл гулять'
```

```
s:='Вася пошёл гулять';
```

или оператор ввода с клавиатуры:

```
ввод s
```

```
readln(s);
```

Обратите внимание, что при вводе строк в Паскале нужно использовать оператор `readln` (англ. *read line* — читать до конца строки) вместо `read`.

Существуют стандартные функции, которые определяют длину строки (количество символов в ней). В школьном алгоритмическом языке такая функция называется `длин`, а в Паскале — `Length` (в переводе с англ. — длина). В следующем примере в целочисленную переменную *n* записывается длина строки *s*:

```
n:=длин(s)
```

```
n:=Length(s);
```

Для того чтобы работать с отдельными символами строки, к ним нужно обращаться так же, как к элементам массива: в квадратных скобках записывают номер символа. Например, так можно изменить четвёртый символ строки[1] на 'а':

```
s[4]:='а'
```

```
s[4]:='a';
```


Приведём полную программу, которая вводит строку с клавиатуры, заменяет в ней все буквы 'а' на буквы 'б' и выводит полученную строку на экран.

```
алг Замена а на б
нач
  лит s
  вывод 'Введите строку: '
  ввод s
  цел i
  нц для i от 1 до длин(s)
    если s[i]='а'
      то s[i]:='б'
    все
  кц
  вывод s
кон
```

```
program ReplaceAB;
var s: string;
    i: integer;
begin
  writeln('Введите строку');
  readln(s);
  for i:=1 to Length(s) do
    if s[i]='a' then
      s[i]:='б';
  writeln(s);
end.
```

Операции со строками

Оператор «+» используется для **объединения (сцепления)** строк, эта операция иногда называется **конкатенацией**. Например:

```
s1:='Привет'
s2:='Вася'
s:=s1 + ', ' + s2 + '!'
```

Здесь и далее считаем, что в программе объявлены **строковые (литерные)** переменные *s*, *s*1 и *s*2. В результате выполнения приведённой программы в строку *s* будет записано значение `'Привет, Вася!'`.

Для того чтобы выделить часть строки (**подстроку**), в школьном алгоритмическом языке применяется операция получения среза[2], например *s*[3:7] означает символы строки *s* с 3-го по 7-й включительно. В Паскале для этого используется функция `Copy`,

[1] Длина строки в этом случае должна быть не менее 4 символов.

[2] В описании школьного алгоритмического языка системы КуМир эта операция называется «вырезка» (англ. *slicing*).

она принимает три аргумента: имя строки, номер начального символа и количество символов. Например, оба следующих фрагмента копируют в строку *s*1 символы строки *s* с 3-го по 7-й (всего 5 символов):

```
s:='123456789'                s:='123456789';
s1:=s[3:7]                    s1:=Copy(s,3,5);
```

В строку *s*1 будет записано значение '34567'.

Для удаления части строки нужно вызвать соответствующую процедуру, указав название строки, номер начального символа и число удаляемых символов:

```
s:='123456789'                s:='123456789';
удалить(s, 3, 6)              Delete(s, 3, 6);
```

В переменной *s* остаётся значение '129' (удаляются 6 символов, начиная с 3-го).

И в школьном алгоритмическом языке, и в Паскале удаление выполняет процедура, которая изменяет переданную ей строку.

При вставке символов соответствующей процедуре передают вставляемый фрагмент, имя исходной строки и номер символа, с которого начинается вставка:

```
s:='123456789'                s:='123456789';
вставить('ABC', s, 3)         Insert('ABC', s, 3);
```

Переменная *s* получит значение '12ABC3456789'.

Поиск в строках

Существуют функции для поиска подстроки (и отдельного символа) в строке. Им нужно передать образец для поиска и строку, в которой надо искать.

```
s:='Здесь был Вася.'          s:='Здесь был Вася.';
n:=позиция('c', s)            n:=Pos('c', s);
если n>0 то                   if n>0 then
  вывод 'Номер символа ', n     write('Номер символа ', n)
иначе                         else
  вывод 'Символ не найден.'     write('Символ не найден.');
все
```


Функция позиция возвращает целое число — номер символа, с которого начинается образец (буква 'c') в строке *s*. Если в строке несколько образцов, функция находит первый из них. Если образец не найден, функция позиция возвращает 0 (недействительный номер символа).

В языке Паскаль функция Pos (от англ. *position* — позиция) работает точно так же.

Пример обработки строк

Предположим, что с клавиатуры вводится строка, содержащая имя, отчество и фамилию человека, например:

```
Василий Алибабаевич Хрюндиков
```

Каждые два слова разделены одним пробелом, в начале строки пробелов нет. В результате обработки должна получиться новая строка, содержащая фамилию и инициалы:

```
Хрюндиков В.А.
```

Возможный алгоритм решения этой задачи может быть написан на псевдокоде так:

```
ввести строку s
найти в строке s первый пробел
имя:=всё, что слева от первого пробела
удалить из строки s имя с пробелом
найти в строке s первый пробел
отчество:=всё, что слева от первого пробела
удалить из строки s отчество с пробелом | осталась фамилия
s:= s + ' ' + имя[1] + '.' + отчество[1] + '.'
```

Мы последовательно выделяем из строки три элемента: имя, отчество и фамилию, используя тот факт, что они разделены одиночными пробелами. После того как имя сохранится в отдельной переменной, в строке *s* останутся только отчество и фамилия. После «изъятия» отчества остаётся только фамилия. Теперь нужно собрать строку-результат из частей: «сцепить» фамилию и первые буквы имени и отчества, поставив пробелы и точки между ними. Для выполнения всех операций будем использовать стандартные функции, описанные выше.

Приведём полные программы на школьном алгоритмическом языке:

```
алг ФИО
нач
  лит s, name, name2
  цел n
  вывод 'Введите имя, отчество и фамилию:'
  ввод s
  n:=позиция(' ', s)
  name:=s[1:n-1]          | вырезать имя
  удалить(s, 1, n)
  n:=позиция(' ', s)
  name2:=s[1:n-1]         | вырезать отчество
  удалить(s, 1, n)        | осталась фамилия
  s:=s + ' ' + name[1] + '. ' + name2[1] + '.'
  вывод s
кон
```

и на Паскале:

```
program FIO;
var s, name, name2: string;
    n: integer;
begin
  write('Введите имя, отчество и фамилию: ');
  readln(s);
  n:=Pos(' ', s);
  name:=Copy(s, 1, n-1);    {вырезать имя}
  Delete(s, 1, n);
  n:=Pos(' ', s);
  name2:=Copy(s, 1, n-1);   {вырезать отчество}
  Delete(s, 1, n);          {осталась фамилия}
  s:=s + ' ' + name[1] + '.' + name2[1] + '.';
  writeln(s)
end.
```

Преобразования число ↔ строка

В практических задачах часто нужно преобразовать число, записанное в виде цепочки символов, в числовое значение, и наоборот. Для этого в школьном алгоритмическом языке есть стандартные функции:

- лит_в_цел — переводит строку в целое число;
- лит_в_вещ — переводит строку в вещественное число;
- цел_в_лит — переводит целое число в строку;
- вещ_в_лит — переводит вещественное число в строку.

Разберём такой пример:

```
лит s, цел N, вещ X, лог OK
s:='123'
N:=лит_в_цел(s, OK) | N = 123
если не OK то вывод 'Ошибка!' все
s:='123.456';
X:=лит_в_вещ(s, OK) | X = 123.456
если не OK то вывод 'Ошибка!' все
```

Строку не всегда можно преобразовать в число (например, если в ней содержатся буквы). Поэтому функции лит_в_цел и лит_в_вещ используют второй аргумент — логическую переменную *OK*, в которую записывается значение да, если операция закончилась успешно, и нет в противном случае. При обратном преобразовании таких проблем быть не может:

```
N:=123
s:=цел_в_лит(N) | s = '123'
X:=123.456
s:=вещ_в_лит(X) | s = '123.456'
```

В языке Паскаль строка преобразуется в число (целое или вещественное) с помощью процедуры Val:

```
var r: integer;
...
s:='123';
Val(s, N, r); {N=123}
if r>0 then writeln('Ошибка!');
s:='123.456';
Val(s, X, r); { X=123.456 }
if r>0 then writeln('Ошибка!');
```

Третий аргумент *r* служит для того, чтобы зафиксировать ошибку: если после вызова процедуры Val он равен нулю, то ошибки не было, иначе в переменную *r* записывается номер первого ошибочного символа.

Преобразование числа в строку выполняет процедура Str:

```
N:=123;
Str(N, s); { s='123' }
X:=123.456;
Str(X, s); { s='1.234560E+002' }
Str(X:10:3, s); { s='   123.456' }
```

По умолчанию вещественные числа представлены в научном (экспоненциальном) формате ('1.234560E+002' означает $1{,}23456 \cdot 10^2$). В последней строке примера используется **форматный вывод**: запись X:10:3 означает «вывести число в 10 позициях с 3 знаками в дробной части».

Строки в процедурах и функциях

Строки можно передавать в процедуры и функции как аргументы (значения параметров), а также возвращать как результат функций. Построим процедуру, которая заменяет в строке *s* все вхождения слова-образца *wOld* на слово-замену *wNew* (здесь *wOld* и *wNew* — это имена переменных, а выражение «слово *wOld*» означает «слово, записанное в переменную *wOld*»).

Сначала разработаем алгоритм решения задачи. На первый взгляд кажется, что можно написать такой алгоритм на псевдокоде:

```
нц пока слово wOld есть в строке s
 удалить слово wOld из строки
 вставить на это место слово wNew
кц
```

Однако такой алгоритм работает неверно, если слово *wOld* входит в состав *wNew*, например, нужно заменить '12' на 'A12B' (покажите самостоятельно, что это приведет к зацикливанию).

Чтобы избежать подобных проблем, попробуем накапливать результат в другой символьной строке *res*, удаляя из строки *s* уже обработанную часть. Предположим, что на некотором шаге в оставшейся части строки *s* обнаружено слово *wOld* (рис. 8.13, *а*).

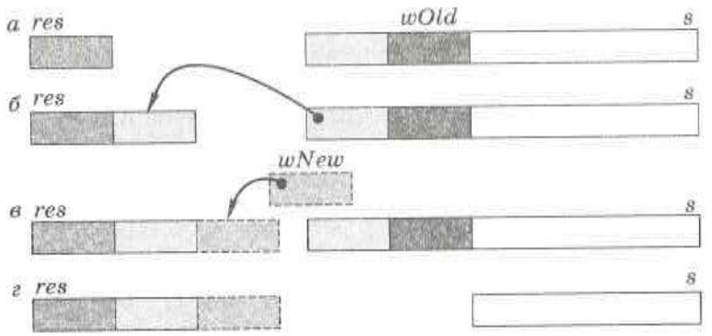


Рис. 8.13

Теперь нужно выполнить следующие действия:

1) ту часть строки *s*, которая стоит слева от образца, «прицепить» в конец строки *res* (рис. 8.13, *б*);
2) «прицепить» в конец строки *res* слово-замену *wNew* (рис. 8.13, *в*);
3) удалить из строки *s* начальную часть, включая найденное слово-образец (рис. 8.13, *г*).

Далее все эти операции (начиная с поиска слова *wOld* в строке *s*) выполняются заново до тех пор, пока строка *s* не станет пустой. Если очередное слово-образец найти не удалось, вся оставшаяся строка *s* приписывается в конец строки-результата, и цикл заканчивается.

В начале работы алгоритма в строку *res* записывается пустая строка '', не содержащая ни одного символа. В таблице 8.3 приведён протокол работы алгоритма замены для строки '12.12.12', в которой нужно заменить слово '12' на 'A12B'.

Таблица 8.3

Рабочая строка *s*	Результат *res*
'12.12.12'	''
'.12.12'	'A12B'
'.12'	'A12B.A12B'
''	'A12B.A12B.A12B'

Теперь можно написать процедуру на школьном алгоритмическом языке. Так как она должна менять строку *s*, эта строка должна быть одновременно аргументом и результатом (в школьном алгоритмическом языке — **аргрез**), а старое и новое слова — это просто аргументы (**арг**).

```
алг replaceAll(аргрез лит s, арг лит wOld, wNew)
нач
  лит res
  цел p, len
  len:=длин(wOld)
  res:=''
  нц пока длин(s)>0
    p:=позиция(wOld, s)
    если p=0 то res:=res+s; выход все
    если p>1 то res:=res+s[1:p-1] все
    res:=res+wNew
    если p+len>длин(s) то
          s:=''
    иначе s:=s[p+len:длин(s)]
    все
  кц
  s:=res
кон
```

Дадим некоторые пояснения к программе. Переменная *p* — это номер первого символа первого найденного слова-образца *wOld*, а в переменной *len* записана длина этого слова. Если после поиска слова значение *p* равно нулю (образец не найден), происходит выход из цикла:

```
если p=0 то res:=res+s; выход все
```

Если $p > 1$, то слева от образца есть какие-то символы, и их нужно «прицепить» к строке *res*:

```
если p>1 то res:=res+s[1:p-1] все
```

Условие `p+len>длин(s)` означает, что образец стоит в самом конце слова, при этом остаток строки *s* — пустая строка.

В конце программы результат записывается на место исходной строки *s*.

Приведём пример использования процедуры:

```
алг Замена всех
нач
 лит s='12.12.12'
 replaceAll(s, '12', 'A12B')
 вывод s
кон
```

Построенную выше процедуру можно легко превратить в функцию. Для этого нужно:

- в заголовке функции указать, что она возвращает строку (добавить ключевое слово **лит**);
- все параметры должны быть аргументами (нужно убрать **аргрез** и **арг**);
- поскольку в школьном алгоритмическом языке нельзя менять аргументы внутри процедуры, назовём первый параметр (исходную строку) *s*0, и введём дополнительную переменную *s* для работы со строкой в процедуре;
- в конце нужно записать результат во встроенную переменную *знач*, а не в *s*.

Ниже показаны все изменённые части подпрограммы:

```
алг лит replaceAll(лит s0, wOld, wNew)
нач
  лит s
  s:=s0
  ...  | тело процедуры
  знач:=res
кон
```

Вызывать функцию можно таким образом:

```
алг Замена всех
нач
  лит s='12.12.12'
  s:=replaceAll(s, '12', 'A12B')
  вывод s
кон
```