

Федеральный государственный образовательный стандарт
Образовательная система «Школа 2100»

А.В. Горячев, В.Г. Герасимова, Л.А. Макарина,
А.В. Паволоцкий, А.А. Семёнов, Т.Л. Чернышёва

ИНФОРМАТИКА

УЧЕБНИК • 8 класс • Книга 2



Рекомендовано Министерством образования и науки
Российской Федерации

Москва
БАЛСС
2013

УДК 373.167.1:004+004(075.3)

ББК 32.81я721

Г67

Федеральный государственный образовательный стандарт
Образовательная система «Школа 2100»

Совет координаторов предметных линий Образовательной системы «Школа 2100» –
лауреат премии Правительства РФ 2008 года в области образования
за теоретическую разработку основ образовательной системы нового поколения
и её практическую реализацию в учебниках

На учебник получены положительные заключения Российской академии наук (от 14.10.2011)
№ 10106-5215/448 и Российской академии образования (от 24.10.2011) № 01-5/7д-126

Руководитель издательской программы –
доктор пед. наук, проф., чл.-корр. РАО Р.Н. Бунеев

Авторский коллектив:

А.В. Гиглавый – научный редактор, А.В. Горячев – автор концепции курса, научный
руководитель, В.Г. Герасимова (книга 1: модуль «Выступление с компьютерным сопро-
вождением»), Л.А. Макарина (книга 1: модуль «Поиск информации»), С.Л. Островский
(книга 1: модуль «Управление личными проектами»), А.В. Паволоцкий (книга 2: модуль
«Алгоритмизация и программирование»), А.А. Семёнов, А.Г. Юдина (книга 1: модуль
«Принятие решений»), Т.Л. Чернышёва (книга 2: модуль «Системы счисления»)

Данный учебник в целом и никакая его часть не могут быть
скопированы без разрешения владельца авторских прав

ISBN 978-5-85939-940-6 (кн. 2)

ISBN 978-5-85939-997-0

© А.В. Горячев, В.Г. Герасимова,
Л.А. Макарина, А.В. Паволоцкий,
А.А. Семёнов, Т.Л. Чернышёва, 2012
© ООО «Баласс», 2012

Дорогие ребята!

Профессии, связанные с развитием компьютерной техники и компьютерных программ, пользуются большим спросом в разных странах мира. Иногда такие специалисты живут в одном городе, а работают в другом или даже в другой стране. Теоретические основы для овладения этими профессиями называются «Компьютерными науками» (Computer Science). Чтобы стать хорошим профессионалом, надо не жалеть сил и времени на их изучение.

Во второй книге учебника мы разместили учебные модули, с помощью которых вы сможете изучить теоретические основы информатики, сквозную линию модулей с 7-го по 9-й класс, нацеленную на обучение программированию, а также модули профессиональной ориентации, с помощью которых можно научиться основам профессий, опирающихся на применение компьютеров.

Во второй книге учебника для 8-го класса, которую вы держите в руках, в модуле «Алгоритмизация и программирование» вы сможете продолжить обучение программированию на языке Паскаль. Модуль «Системы счисления» относится к теоретическим модулям, с его помощью можно узнать про разные системы счисления, научиться переводить числа из одной системы счисления в другую и выполнять над ними арифметические действия в разных системах счисления.

Как работать с учебником

Просмотрите «Содержание», перелистайте учебник. Вы заметите, что он разделён на модули. Вы будете изучать модули в том порядке, который предложит учитель.



Практически в каждом модуле мы предусмотрели пять основных параграфов. Изучив эти параграфы, вы напишете проверочную работу, по итогам которой узнаете, как вы освоили новый материал: ниже необходимого уровня, на необходимом или повышенном уровне. Далее вы будете работать самостоятельно, выполняя по указанию учителя задания того уровня, которого вы пока не достигли. Если проверочная работа покажет, что вы освоили и повышенный уровень, то вы будете выполнять задания самого высокого – максимального уровня. Учитель в любой момент может предложить вам перейти на выполнение заданий более высокого уровня. По окончании выполнения заданий учебника учитель проведёт итоговую проверочную работу.



Далее в модуле расположены дополнительные параграфы, задания к ним и проверочные работы. Основные параграфы выделены в учебнике зелёной полосой вверху страницы, дополнительные – розовой полосой.

Дополнительный материал, который вы не изучите на уроках, вы сможете использовать на факультативах и кружках.

На уроках информатики вы сможете освоить умения, которые помогут вам более эффективно использовать компьютеры и компьютерные сети для решения возникающих в вашей жизни задач. Кроме того, учитель может решить, что вам надо освоить умения, которые помогут вам заниматься разработкой новых компьютерных программ или заложат основы профессиональной деятельности, тесно связанной с применением компьютерной техники.

Кроме того, наш учебник, как и все учебники Образовательной системы «Школа 2100», поможет вам в развитии универсальных учебных действий. В учебнике вам могут встретиться задания, обозначенные кружками и фоном разного цвета – это условные знаки. Каждый цвет соответствует определённой группе умений:

-  – организовывать свои действия: ставить цель, планировать работу, действовать по плану, оценивать результат;
-  – работать с информацией: самостоятельно находить, осмысливать и использовать её;

-  –общаться и взаимодействовать с другими людьми, владеть устной и письменной речью, понимать других, договариваться, сотрудничать;
-  –развивать качества своей личности, оценивать свои и чужие слова и поступки;



так обозначены задания, где нужно применить разные группы умений, мы называем их жизненными задачами и проектами.

Для успешного изучения информатики и овладения универсальными учебными действиями на уроках используется проблемно–диалогическая образовательная технология. Поэтому структура параграфа, где вводится новый материал, имеет в учебнике следующий вид.

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Это подведение к теме (вопросу, цели) урока: вы обсуждаете проблему в предложенном материале и формулируете главный вопрос урока (всем классом, в группе или в паре). Сравните свой вариант вопроса с авторским. Авторские вопросы к параграфам расположены в «Содержании» под названиями параграфов и выделены курсивом.

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Так обозначены вопросы и задания по изученному материалу, который вам необходим для открытия нового знания.

РЕШЕНИЕ ПРОБЛЕМЫ

Вы в группе, в паре или совместно с учителем, ведя диалог, осуществляете поиск решения проблемы. Для решения проблемы вы работаете с текстом.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

На этом этапе вы формулируете вывод и проверяете свои предположения, сравнивая их с авторским решением проблемы – научными формулировками правил или определений.

ПРИМЕНЕНИЕ ЗНАНИЙ

Так обозначены задания на применение новых знаний.

ОПЕРАЦИИ

Раздел «Операции» позволит вам научиться выполнять действия с компьютерными программами, необходимые для решения учебных задач.

В конце модуля вы найдёте раздел «Решаем жизненные задачи и работаем над проектами». Задачи и проекты могут выполняться как на уроках, так и на факультативах и кружках.

Там же находится очень важный раздел «О профессиях». Прочитайте его и подумайте, какие профессии вам больше по душе.

Что такое жизненные задачи?

Это проблемы, с которыми вы можете столкнуться в жизни и для решения которых вам понадобятся разные знания и умения. Они оформлены следующим образом:

Название задачи

Ваша роль: человек, в роли которого вы должны себя представить, решая проблему.

Описание. Условия, в которых возникла проблема.

Задание. То, что нужно сделать и получить в итоге.

Что такое проект?

Это любое самостоятельное дело, которое предполагает:

- 1) оригинальный замысел (цель);
- 2) выполнение работы за определённый отрезок времени;
- 3) конкретный результат, представленный в итоге.

Что можно считать результатом проекта?

- Предметы, сделанные своими руками: макеты, модели или вещи для практического использования.
- Мероприятия: спектакли, фотовыставки, викторины, конференции, праздники и тому подобное – при условии, что они подготовлены самими учениками.
- Информационные продукты: газеты, книжки, плакаты, карты, стихотворения, рассказы, доклады, отчёты об исследованиях и т. д.
- Решение конкретных проблем: изменение, улучшение конкретной ситуации, например уборка мусора на школьном дворе.

Правила проектной деятельности

1. Каждый может начать собственный проект.
2. Каждый может объединиться с другими в ходе работы над проектом.
3. Каждый может выйти из проекта при условии, что он не подводит других.
4. Каждый может не участвовать ни в одном проекте.

Как оценить свои учебные достижения?

Для этого надо освоить алгоритм самооценки:

1. Какова была цель задания (что нужно было получить в результате)?
2. Вы выполнили задание (получен ли результат)?
3. Вы выполнили задание верно или с ошибкой?
4. Вы выполнили задание самостоятельно или с чьей-то помощью?
5. Вспомните, как вы ставите отметки. Определите свою отметку.



Модуль 1. Алгоритмизация и программирование

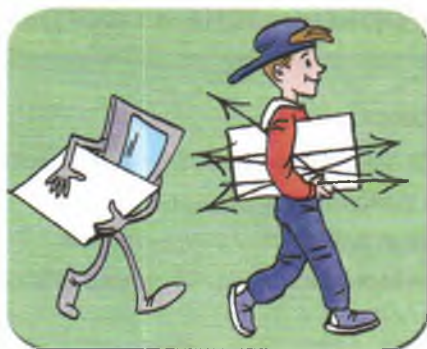
Этот модуль поможет вам:

- познакомиться с основами математической логики;
- изучить методы поиска информации;
- исследовать новые алгоритмы и технологии;
- разобраться в новых возможностях языка Паскаль.

Для этого вам надо научиться:

- составлять программу на языке Паскаль;
- использовать основные конструкции и типы данных языка Паскаль;
- работать в интегрированной среде разработки программ;
- искать ошибки в программах – отлаживать их.

Введение



Дорогие друзья!

В 7-м классе мы с вами немного погрузились в мир программирования, научились писать программы на языке Паскаль, начали работать в интегрированной среде разработки программ, узнали некоторые важные алгоритмы.

В этом году вам предстоит развивать свои знания о программировании и об устройстве компьютера. Кроме того, мы продолжим знакомить вас с программированием на языке Паскаль и различными алгоритмами.

Помните, что единственный способ стать хорошим программистом – это практика, практика и ещё раз практика.

Удачи вам!

§ 1. Знакомство с математической логикой

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Вы читали книгу «Алиса в стране чудес» Льюиса Кэрролла? А знаете ли вы, что Льюис Кэрролл – известный английский математик? Прочитайте его высказывание относительно темы, которую мы будем рассматривать далее:

«Методы эти позволяют Вам обрести ясность мысли, способность находить собственное оригинальное решение трудных задач, вырабатывают у Вас привычку к систематическому мышлению и, что особенно ценно, умение обнаруживать логические ошибки, изъяны и пробелы тех, кто не пытался овладеть привлекательным искусством логики. Попробуйте. Вот всё, о чём я прошу Вас».

- Как вы считаете, о методах какой науки говорит Льюис Кэрролл? Сформулируйте тему урока. Сравните свою формулировку с авторской (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните свои умения программирования на языке Паскаль. (Учебник для 7-го класса, книга 2, модуль 1.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7-го класса, книга 2, модуль 1.)

РЕШЕНИЕ ПРОБЛЕМЫ

ЧТО ТАКОЕ ЛОГИКА?

Ещё в древние времена **логикой** называли умение строго и чётко строить рассуждения, доказывать свои убеждения, выигрывать споры и соревнования. Основу логики как науки заложил великий древнегреческий учёный **Аристотель**.

Рассуждая, люди выражают свои мысли с помощью высказываний. Что такое высказывание? Рассмотрим повествовательное предложение: «*Сейчас идёт дождь*». Выглянув на улицу, вы точно можете определить, идёт сейчас дождь или нет, то есть однозначно сказать, истинно или ложно это предложение. Есть всего два варианта: либо дождь идёт, либо нет. Задав вопрос: «Сейчас идёт дождь?» – вы получите один из двух ответов: «да» или «нет». Третьего не дано.

Повествовательное предложение, относительно которого можно сказать, истинно оно или ложно, в логике называется **высказыванием**.

Вот примеры высказываний:

«Волга впадает в Каспийское море».

«Дважды два равно пять».

«Медведи не летают».

Первое и третье из этих высказываний истинны, а второе – ложно.

Вопросительные и побудительные предложения высказываниями не являются.

Итак, любое высказывание является истинным или ложным. Значения «истина» и «ложь» называются **логическими значениями**.

Высказывания могут быть заданы как словесно, так и формально – с помощью букв. Например, можно обозначить буквой A высказывание «Волга впадает в Каспийское море». Кроме того, высказывания можно задать и на языке программирования. Например, выражение $a = b$ – это высказывание, значение которого может быть или «истина», или «ложь», в зависимости от значений переменных a и b . Формальную запись высказывания в языке программирования называют **логическим выражением**.



Джордж Буль
(1815–1864)

Раздел математики, изучающий рассуждения, доказательства с помощью математических методов, называется **математической логикой**. Рассуждения в математической логике изучаются с точки зрения формы, а не смысла. Одним из основателей математической логики считается известный английский математик и логик **Джордж Буль**.

Алгебра логики (алгебра высказываний) – это раздел математической логики, изучающий высказывания и операции над ними. Алгебру логики ещё называют **булевой алгеброй** – по имени Дж. Буля.

Аппарат алгебры логики применяется при разработке и описании функционирования электронных схем элементов компьютера, работа которых основана на двух устойчивых состояниях переключателей: «включено» (ток идёт)/«выключено» (тока нет), «да»/«нет», «истина»/«ложь», 0/1. Как вы знаете, данные в компьютере представлены последовательностями двух значений – 0 и 1.

ЛОГИЧЕСКИЙ ТИП И ЛОГИЧЕСКИЕ ПЕРЕМЕННЫЕ В ПАСКАЛЕ

Логические значения надо как-то хранить в памяти компьютера. Вы уже знакомы с языком Паскаль и знаете, что для хранения значений необходимы **переменные**, а переменные всегда должны иметь какой-то определённый тип.

Для хранения логических значений в языке Паскаль существует тип, который называется *boolean* (по имени Дж. Буля). Соответственно, имея тип, мы можем описывать переменные этого типа в разделе **var** нашей программы:

```
var
  f: boolean;
```

Переменная *f*, которую мы описали, может принимать всего два значения: «истина» и «ложь». Эти значения представлены в языке Паскаль английскими словами *true* (истина) и *false* (ложь). Таким образом, возможны операции:

```
f := true;
f := false;
if (f = true) then a := 5;
while (f = false) do
  begin
    ***
  end;
```

Вспомните, что при выполнении условного оператора **if** или оператора цикла с предусловием **while...do** проверяется, является ли значение условия истинным. Поэтому выражение *f* = *true* можно не писать, а вместо него оставить только *f*, так как здесь переменная *f* и является условием, проверяется её логическое значение:

```
if f then a := 5
```

Основное назначение **логических переменных** — принимать и хранить значения логических выражений. Например, после выполнения оператора *f* := (1 = 2) в переменной *f* окажется значение *false*, поскольку 1 не равно 2. А после выполнения оператора *f* := (1 < 2) переменная *f* примет значение *true*, поскольку 1 меньше 2. Конечно же, вместо чисел в выражениях могут стоять переменные. Например:

```
f := (a = b)
```

Логические переменные можно сравнивать, но к ним неприменимы арифметические операции. Сравнение логических переменных основано на том соглашении, что *false* меньше *true*.

Значения логических переменных можно выводить с помощью процедуры *writeln*. При этом будет выведено словесное значение переменной: *True* или *False*.

Кроме того, к логическим переменным применимы логические операции, которые мы сейчас с вами и рассмотрим.

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

До этого момента мы рассматривали **простые высказывания**. Сложные высказывания состояются из нескольких простых. Например, высказывание «Я сегодня вечером пойду в кино или пойду в театр» состоит из двух простых:

- 1) «Я сегодня вечером пойду в кино»;
- 2) «Я сегодня вечером пойду в театр»

и специальной *связки* – союза «или».

Таким связкам соответствуют **логические операции**.

Рассмотрим три наиболее важные операции, которые присутствуют в языке Паскаль.

ОПЕРАЦИЯ НЕ

Первая логическая операция называется **НЕ**. В языке Паскаль она обозначается английским словом **not**.

Дадим определение операции НЕ.

Определение. Если применить операцию НЕ к высказыванию, то его значение изменится на противоположное. Если оно было истинным, то станет ложным, а если было ложным, то превратится в истинное.

Вспомните рассмотренный выше фрагмент программы: там можно оператор цикла с предусловием

```
while (f = false) do
```

заменить на

```
while (not f) do
```

Здесь проверяется значение условия **not f**: если оно истинно (то есть значение **f** ложно), то цикл продолжает выполняться.

Операция НЕ – *унарная* (от латинского слова *unus* – единственный), то есть она применяется к одному высказыванию. Объект, над которым совершается операция, называется **операндом**.

Рассмотрим простую программу.

Код программы	Вывод
<pre> program logic1; var f: boolean; begin f := false; writeln('До операции not: ', f); f := not false; writeln('После операции not: ', f); end.</pre>	<p>До операции not: False После операции not: True</p>

Как видно из примера, значение переменной *f* после выполнения операции **not** изменилось с *false* на *true*.

ОПЕРАЦИЯ И

Следующая логическая операция называется **И**. В языке Паскаль она обозначается английским словом **and**. Этой операции соответствует связка «и» или «а». Примеры высказываний: «Сейчас холодно и идёт снег»; «На дворе ветер, а всё равно тепло».

Определение. Значение операции И – «истина» тогда и только тогда, когда оба простых высказывания, которые она объединяет, истинны.

Примеры:

- 1) Логическое выражение $(2 > 1)$ **and** $(5 = 10 \text{ div } 2)$ истинно, поскольку $2 > 1$, а 5 равно 10, делённому нацело на 2.
- 2) Логическое выражение $(1 = 2)$ **and** $(5 > 0)$ ложно, поскольку хотя 5 больше нуля, но 1 не равно 2.

Операция И называется бинарной – она применяется к двум операндам.

Давайте напишем программу, которая будет выводить все возможные комбинации значений операции И.

Код программы	Вывод															
<pre>program logic2; var a, b: boolean; begin writeln(' a ', ' b ', ' a И b'); writeln('-----'); for a := false to true do for b := false to true do writeln(a:7, b:7, (a and b):7); end. end.</pre>	<table><tr><th>a</th><th>b</th><th>a И b</th></tr><tr><td>False</td><td>False</td><td>False</td></tr><tr><td>False</td><td>True</td><td>False</td></tr><tr><td>True</td><td>False</td><td>False</td></tr><tr><td>True</td><td>True</td><td>True</td></tr></table>	a	b	a И b	False	False	False	False	True	False	True	False	False	True	True	True
a	b	a И b														
False	False	False														
False	True	False														
True	False	False														
True	True	True														

В нашем примере операнды – это *a* и *b*.

Как мы видим, существует всего четыре варианта. Кроме того, пример показывает нам, что логические переменные можно использовать в цикле (поскольку *false* всегда меньше *true*), а также логические переменные можно *форматировать* при выводе.

Таблица, которую мы с вами получили в итоге, называется **таблицей истинности** логической операции. Посмотрите на неё внимательно. Когда операция И истинна? Когда и *a*, и *b* истинны, и только в этом случае.

- Сравните полученный результат с определением операции И.

ОПЕРАЦИЯ ИЛИ

Третья логическая операция, которую мы с вами рассмотрим, называется **ИЛИ**. Её обозначение в языке Паскаль – **or**.

Определение. Значение операции ИЛИ – «истина» в том случае, когда *хотя бы одно* из высказываний, входящих в её состав, истинно, и «ложь» – в противном случае.

Примеры:

- 1) Логическое выражение $(1 = 2) \text{ or } (5 < 10)$ истинно, поскольку хотя 1 не равно 2, но 5 меньше 10.
- 2) Сложное логическое выражение $(1 = 2) \text{ or } (5 < 0)$ ложно, поскольку оба простых логических выражения ложны.

Операция ИЛИ, так же как и операция И, является *бинарной*, поскольку у неё есть два операнда.

Давайте получим таблицу истинности операции ИЛИ.

Код программы	Вывод
<pre> program logic3; var a, b: boolean; begin writeln(' a ', ' b ', ' a ИЛИ b'); writeln('-----'); for a := false to true do for b := false to true do writeln(a:7, b:7, (a or b):7); end. </pre>	<pre> a b a ИЛИ b ----- False False False False True True True False True True True True </pre>

Как мы видим, операция ИЛИ ложна только в том случае, когда оба её операнда ложны, во всех остальных случаях операция истинна.

ПРИОРИТЕТЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Как и в случае с арифметическими операциями, логические операции имеют приоритеты по выполнению. Рассмотрим логическое выражение: `a or not b and c`. Никаких скобок в нём нет, какая же операция будет выполняться первой, а какая – второй?

Самым высшим приоритетом, то есть операцией, которая будет выполняться первой, обладает операция НЕ, затем идёт операция И, а самый низкий приоритет имеет операция ИЛИ.

Таким образом, в нашем примере сначала будет вычислено `not b`, затем `not b and c`, а потом к результату будет применена операция `or`.

ПРИМЕНЕНИЕ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Рассмотрим задачу, в которой продемонстрируем применение логических операций.

Задача. Последовательность чисел вводится с клавиатуры до нулевого значения. Необходимо определить, является ли она возрастающей, то есть верно ли утверждение, что каждое следующее число больше предыдущего. Ноль учитывать не надо.

Пример возрастающей последовательности: 1, 3, 5, 7, 8, 10.

Пример невозрастающей последовательности: 1, 2, 3, 3, 2, 1.

```
program logic4;
var
  a, b: integer;
  f: boolean;
begin
  writeln('Вводите числа до нуля');
  // Предположим, что последовательность
  // изначально возрастающая
  f := true;
  readln(a);
  // будем продолжать цикл, пока a не ноль
  while (a <> 0) do
    begin
      // Введём новое число b
      readln(b);
      // Проверим, не нарушилось ли возрастание
      // с учётом предыдущих значений.
      // Поскольку 0 учитывать не надо, проверим b на
      равенство нулю
      if (b <> 0) then f := f and (b > a);
```

```
// сохраним b в a
a := b;
end;
// Если f = true, то возрастание не нарушилось
if f then writeln('Последовательность возрастающая')
else writeln('Последовательность невозрастающая');
end.
```

Комментарии, указанные в тексте программы, объясняют её работу.

Обратите внимание на выделенные фрагменты. Логическая переменная *f* является признаком того, какая последовательность была ранее, до текущего момента – момента проверки. Мы предположили, что наша последовательность возрастающая, и занесли в *f* начальное значение *true*. Затем, каждый раз (в цикле), вводя *b*, мы выполняем операцию присваивания *f* := *f* **and** (*b* > *a*). Если значение *f* было равно *false*, то в каком бы отношении ни находились переменные *a* и *b*, *f* своё значение в дальнейшем не поменяет (вспомните операцию И). Это означает, что если в последовательности попался хотя бы один элемент, нарушающий её возрастание, то она так и останется невозрастающей, независимо от того, какие элементы к ней добавятся. Если же значение *f* было равно *true*, и на текущем шаге *b* станет меньше или равным *a*, то *f* изменит своё значение на *false* (нашёлся элемент, нарушающий возрастание).

Как вы думаете, что бы вы получили, если бы не использовали операцию И, а написали бы просто *f* := (*b* > *a*) ? В *f* остался бы результат *последнего* сравнения. А нам этого мало, поскольку мы должны проанализировать *всю* последовательность.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Раздел математики, изучающий рассуждения, доказательства с помощью математических методов, называется математической логикой.

Алгебра логики (алгебра высказываний) – это раздел математической логики, изучающий высказывания и операции над ними.

Высказывание – это повествовательное предложение, относительно которого можно сказать, истинно оно или ложно. Высказывания бывают простыми и сложными. Значения «истина» и «ложь» называются логическими значениями.

Формальную запись высказывания в языке программирования называют логическим выражением.

Для описания логических значений в языке Паскаль имеется тип *boolean*. Логические значения записываются как *true* и *false*. Логические переменные можно сравнивать, но к ним не применимы арифметические операции.

Сложные высказывания получаются из простых с помощью логических операций: И, ИЛИ, НЕ. В языке Паскаль им соответствуют операторы *and*, *or*, *not*.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Найдите значение логической переменной.

а) $f := (5 \bmod 2 = 0)$

б) $f := (9 \operatorname{div} 3 > 2) \text{ and } (15 \bmod 2 = 0)$

в) $f := \text{not } (5 \operatorname{div} 2 = 0) \text{ or } (2 > 0) \text{ and } (48 > 96 \operatorname{div} 2)$

2. Из приведённых предложений выделите высказывания. Объясните свой выбор.

а) *Который сейчас час?*

б) *Сегодня жаркая погода, а вчера было холодно.*

в) *Моему другу Васе 13 лет.*

г) *Собирайся в школу!*

д) *Мы в школе изучаем информатику.*

3. Напишите программу, которая вводит с клавиатуры два действительных числа A и B , удваивает эти числа, если $A \geq B$, в противном случае заменяет их противоположными значениями. Программа должна выводить полученные значения.

§ 2. Поиск в массиве

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Учитель биологии разложил на столе перед учениками карточки с изображениями различных зверей картинками вниз, так что ребята самих изображений не видели.

– Найдите, пожалуйста, среди этих карточек ту, на которой изображён заяц, – сказал учитель.

– Но ведь мы не видим, что изображено на карточках, – ответила Маша, – и не знаем, есть ли вообще там нужная картинка!

• Как можно помочь Маше? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Какие существуют способы записи алгоритмов? (Учебник для 7–го класса, книга 2, модуль 1, § 2.)

Вспомните свои навыки работы с массивами. (Учебник для 7–го класса, книга 2, модуль 1, § 8–9.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7–го класса, книга 2, модуль 1.)

Что такое отладка программ и как она выполняется? (Учебник для 7–го класса, книга 2, модуль 1, § 7.)

РЕШЕНИЕ ПРОБЛЕМЫ

Вы уже научились заносить данные в массив, обрабатывать их, выводить на экран, одним словом, работать с содержимым массива. Продолжим изучать алгоритмы работы с массивами – попробуем решить задачу **поиска элемента** в массиве.

Будем рассматривать алгоритмы поиска на примере целочисленных массивов.

ПРОСТОЙ ЛИНЕЙНЫЙ ПОИСК

Представьте, что у вас есть непрозрачный мешок, в котором лежат пронумерованные шарики. И вы хотите вытащить из этого мешка шарик с номером 5, причём вы не знаете, есть там такой шарик или нет. Как вы поступите?

Скорее всего, вы будете по одному вытаскивать шарики из мешка и смотреть на их номера. Если вы вытащите шарик с желаемым номером 5, то ваша

задача будет решена и поиск можно будет считать завершившимся удачно. Однако может случиться и так, что вы вытащите все шарики, а нужного среди них не окажется. Эта ситуация говорит о том, что шарика с номером 5 в мешке не было.

Давайте попробуем решить нашу задачу с помощью компьютера.

Мешок с шариками представим в виде массива целых чисел. Формулировка задачи поиска будет звучать следующим образом: найти в массиве элемент с требуемым значением; если его там нет, вывести сообщение об этом.

Запишем алгоритм в виде блок-схемы (рис. 1.1) и на языке программирования Паскаль.

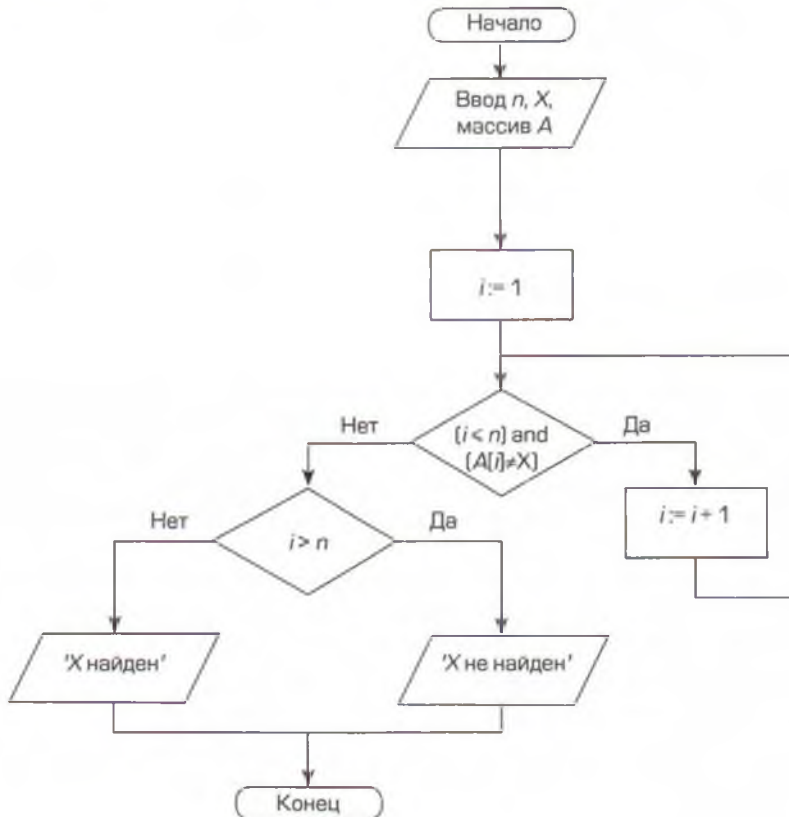


Рис. 1.1

```

program search1;
var
  A: array [1..100] of integer;
  x, n, i: integer;

```

begin

```
writeln('Поиск числа в массиве');
// Ввод массива
write('Введите количество элементов в массиве: ');
readln(n);
for i := 1 to n do
begin
    write('A[', i, ']=');
    readln(A[i]);
end;
// Конец ввода массива
// Ввод искомого числа
write('Введите искомое число: ');
readln(x);
// Поиск
i := 1;
while ((i <= n) and (x <> A[i])) do i := i + 1;
if (i > n)
    then writeln('Число ', x, ' в массиве не найдено')
    else writeln('Число ', x, ' в массиве найдено');
end.
```

В начале алгоритма происходит ввод количества элементов массива — n и заполнение массива A . Заполнение происходит с помощью цикла с параметром **for**. Затем вводится искомое число x . Поиск выполняется с помощью цикла с предусловием. Мы начинаем просматривать массив с первого элемента ($i := 1$). Условием продолжения цикла является логическое выражение $((i \leq n) \text{ and } (x \neq A[i]))$. Цикл можно описать так: пока i не превысило n (то есть пока мы не перебрали все элементы массива) и искомое число x не равно текущему элементу массива $A[i]$, будем увеличивать i .

У нас только два варианта прекращения этого цикла:

- 1) мы просмотрим все элементы (i станет больше n);
- 2) мы найдём искомое число в массиве ($A[i]$ станет равным x).

Проверим это в последнем условном операторе. Если мы просмотрели все элементы, то искомого числа в массиве нет, в противном случае число в массиве присутствует.

Приведённый **поиск** называется **линейным**, поскольку его суть состоит в последовательном (линейном) переборе всех элементов массива до момента принятия решения о наличии или отсутствии в нём искомого числа. Если оно находится на первом месте в массиве, то алгоритм отработает очень быстро, а вот если оно находится в конце массива или отсутствует, то, пока мы не просмотрим весь массив, ответить на вопрос мы не сможем.

В этой тонкости заключается основной недостаток линейного поиска – непредсказуемость времени его работы, но для осуществления поиска в массиве, о структуре которого мы ничего не знаем, у нас нет других вариантов.

Стоит отметить также, что в нашем случае мы искали элемент строго по одному условию – равенству x его значения. Мы можем усложнить критерий поиска. Например, узнать, есть ли в массиве элемент, значение которого равно x , а индекс кратен 5.

- Как надо модифицировать алгоритм, чтобы вы могли получить информацию об индексе искомого элемента массива?
- Представьте, что искомый элемент встречается в массиве дважды. Какой индекс найдет модифицированный алгоритм?

ЛИНЕЙНЫЙ ПОИСК С БАРЬЕРОМ

Предыдущий алгоритм в своей работе использовал сложное логическое выражение в качестве условия продолжения цикла поиска. Сейчас мы рассмотрим механизм, который позволит нам избавиться от этого сложного условия.

Назовём **барьером** элемент массива, равный искомому и добавленный в массив после самого последнего элемента. Таким образом, исходный массив увеличится на 1 элемент, а элемент-барьер будет иметь индекс $n + 1$.

Примеры:

- 1) Массив: 1, 2, 3, 4, 5. Ищем число 3. Массив с барьером: 1, 2, 3, 4, 5, **3**.
- 2) Массив: 3, 2. Ищем число 1. Массив с барьером: 3, 2, **1**.

Теперь вы будете искать число не в исходном массиве, а в массиве с барьером. И что самое важное – вы этот элемент *всегда* найдёте! Вы же его сами добавили в конец массива.

Как в этом случае понять, нашли вы элемент или нет? Вам поможет барьер. Если при поиске элемента вы получили номер $n + 1$, то в исходном массиве искомого числа нет, в противном случае оно есть.

Посмотрите, как изменилась блок-схема алгоритма (рис. 1.2).

При реализации этого алгоритма на языке программирования нужно не забыть увеличить размер массива на 1, чтобы было где разместить барьерный элемент.

Линейный поиск с барьером обладает тем же недостатком, что и простой линейный поиск, – невозможностью предсказания времени работы алгоритма.

Для того чтобы эффективно искать информацию в массивах данных, необходимо приводить массивы к такому виду, при котором становятся известны закономерности размещения в них элементов. Такие алгоритмы называются упорядочением массива по некоторому признаку. Их мы рассмотрим в следующем параграфе.

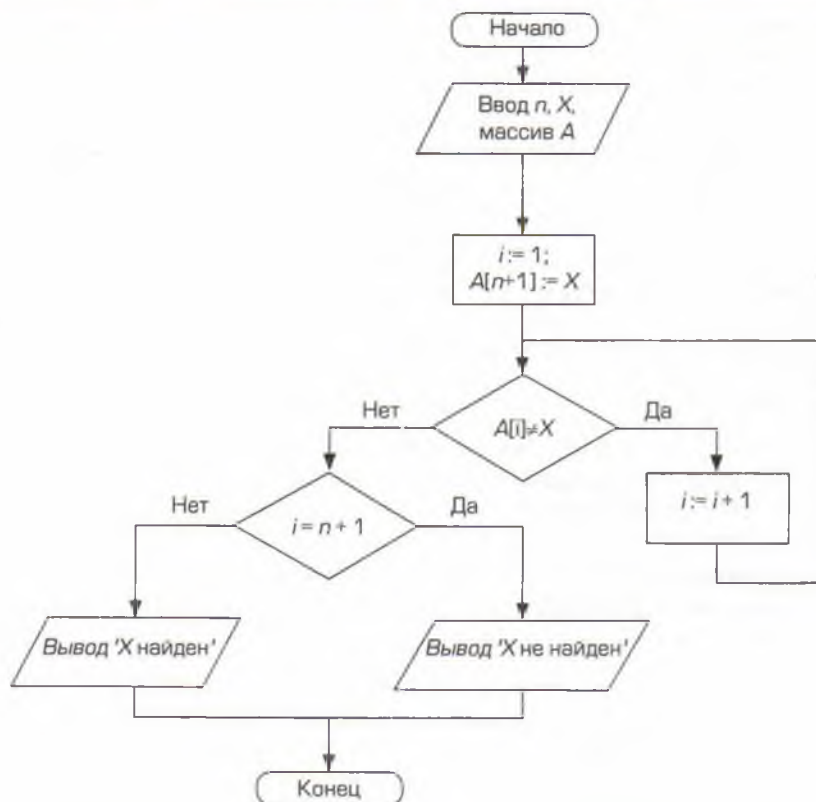


Рис. 1.2

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Чтобы проверить массив на наличие в нём некоторого элемента, необходимо применить алгоритмы поиска элементов.

Алгоритмы линейного поиска (простого и с барьером) позволяют искать данные в произвольном массиве, последовательно перебирая все его элементы.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. По блок-схеме, приведённой в параграфе, напишите на языке Паскаль программу, осуществляющую линейный поиск в целочисленном массиве с барьером.

2. С клавиатуры вводится число n , а за ним – массив из n целых чисел. Найдите номер первого по счёту элемента массива с нечётным значением.

3. С клавиатуры вводится число n , за ним массив из n целых чисел и число x . Найдите количество элементов массива, равных x .

§ 3. Упорядочение массивов

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Перед началом урока физкультуры ученики построились в спортивном зале. Но учителю построение не понравилось.

Ну-ка,стройтесь по росту! – скомандовал учитель.

А как? По возрастанию или по убыванию? – спросил Вася.

• Что не понравилось учителю? О каких правилах построения говорит Вася? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните свои навыки работы с массивами. (Учебник для 7-го класса, книга 2, модуль 1, § 8–9.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7-го класса, книга 2, модуль 1.)

РЕШЕНИЕ ПРОБЛЕМЫ

УПОРЯДОЧЕНИЕ

Порядок – это расположение вещей, людей и любых других объектов, удовлетворяющее заранее определённой правилу.

Например, расположение книг в библиотеке подчиняется правилу алфавитного каталога. Это означает, что книги так расставляются в шкафах, что фамилии их авторов расположены по алфавиту: Петров после Иванова, Иванов после Батюшкова, а Батюшков после Абрикосова. Представьте, как долго вы занимались бы поиском в большой библиотеке, если бы книги там были расставлены случайным образом.

Про информацию, организованную в соответствии с некоторым порядком, говорят, что она **упорядочена**. В упорядоченной информации гораздо проще искать нужные сведения.

В компьютере информация закодирована в виде данных. Поиск в больших массивах данных значительно облегчается при их упорядочении и становится эффективнее. Давайте научимся приводить имеющиеся у нас в компьютере данные к упорядоченному виду.

Упорядочение массивов часто называют **сортировкой**. Алгоритмов сортировки очень много, причём существуют как универсальные алгоритмы, так и алгоритмы, которые применяются в уникальных ситуациях. Мы рассмотрим два алгоритма: сортировку выбором и сортировку обменом.

СОРТИРОВКА ВЫБОРОМ

Пусть у нас имеется произвольный массив целых чисел. Его можно отсортировать **по возрастанию**. Это означает, что каждый следующий элемент массива должен быть строго больше предыдущего. При сортировке **по убыванию**, напротив, каждый следующий элемент массива должен быть строго меньше предыдущего.

Если в массиве имеются одинаковые элементы, то его сортируют **по неубыванию** (следующий элемент больше или равен предыдущему) или **по невозрастанию** (следующий элемент меньше или равен предыдущему).

Например, массив 1 3 5 5 8 9 118 335 отсортирован по неубыванию, а массив 1 3 5 3 5 2 9 10 – нет.

Рассмотрим сортировку по неубыванию.

Пусть исходный массив выглядит так:

3 8 -1 5 2 15 10

Найдём в этом массиве минимальный элемент – это будет -1. Поменяем местами первый элемент массива и найденный нами минимальный. Получим вот такую картину (элементы, которые мы меняли местами, выделены красным цветом):

-1 8 3 5 2 15 10

Теперь мы точно знаем, что на первом месте в массиве стоит самое маленькое число. С этим числом мы больше работать не будем – оно заняло своё место (выделим его на рисунке синим цветом). Вновь поищем в массиве, но уже без первого элемента, самое маленькое число. Мы найдём число 2. Поменяем местами второй элемент массива и найденное число 2. Получим такой массив:

-1 | 2 3 5 8 15 10

Два самых маленьких числа встали в начало массива: -1 2. Их больше не рассматриваем. Будем дальше повторять наши действия. Начиная с третьего элемента (числа 3), будем искать минимальный элемент в массиве. Но третий элемент и есть наименьший, поэтому никакого обмена производить не будем. Получим:

-1 2 | 3 5 8 15 10

Будем продолжать и остановимся в тот момент, когда дойдём до последнего элемента массива:

-1 2 3 | 5 8 15 10
 -1 2 3 5 | 8 15 10
 -1 2 3 5 8 | 10 15
 -1 2 3 5 8 10 | 15
 -1 2 3 5 8 10 15

В результате, как только мы дойдём до последнего элемента массива, мы получим отсортированный массив.

Посмотрите на блок-схему нашего алгоритма (рис 1.3). Синей пунктирной рамкой отмечен фрагмент, в котором происходит поиск минимального элемента, красной – фрагмент, в котором происходит обмен.

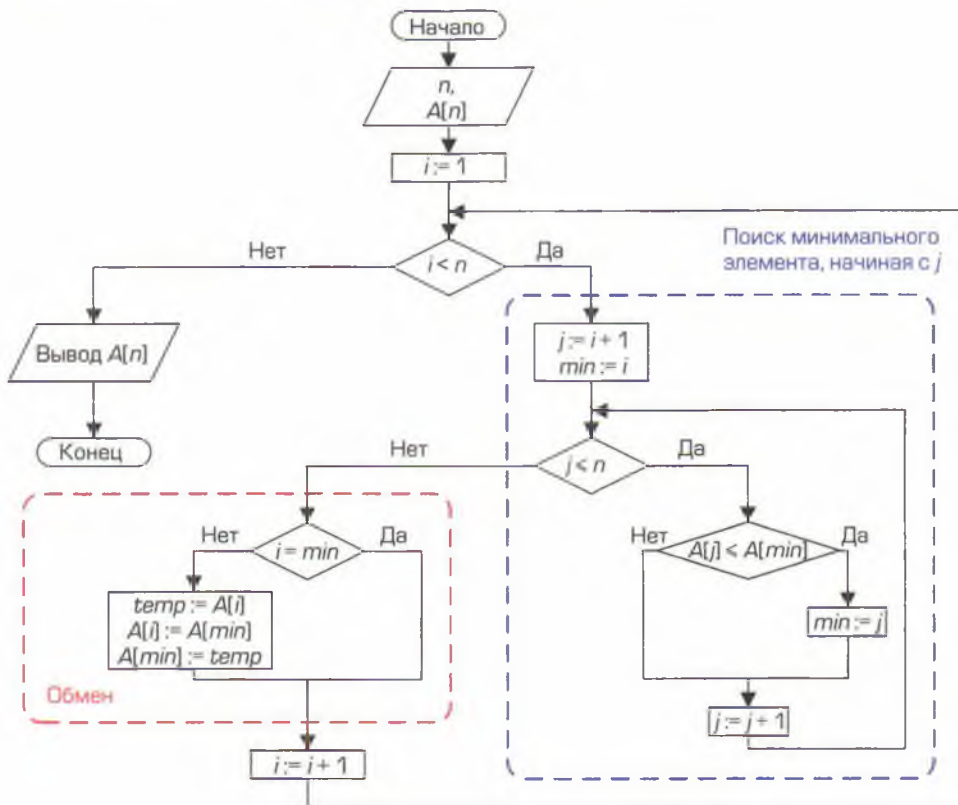


Рис. 1.3

В переменной i хранится номер текущего элемента массива. Часть массива до i -го номера уже отсортирована. Первоначально в переменную i мы заносим значение 1.

Если внимательно проанализировать приведённую блок-схему, то можно увидеть, что она содержит два цикла, один из которых находится внутри другого. Такие циклы называются **вложенными**.

Обратите внимание на условие продолжения первого (внешнего) цикла: $i < n$. Оно говорит нам о том, что последний элемент массива мы не будем рассматривать. Почему? Потому, что массив из одного элемента всегда является отсортированным, и никаких действий для упорядочения применять не надо. Как только мы доберёмся до последнего элемента массива, мы его выводим. На этом алгоритм заканчивается.

Задачу поиска минимального элемента в массиве мы с вами разбирали в 7-м классе. Посмотрите на блок-схему и вспомните её. После выполнения вложенного (внутреннего) цикла в переменной *min* будет находиться номер минимального элемента.

Теперь рассмотрим красный блок. Если номер минимального элемента отличается от *i*, то необходимо произвести обмен. Для обмена нам понадобится ещё одна переменная – *temp*.

СОРТИРОВКА ОБМЕНОМ

Рассмотрим тот же исходный массив:

3 8 -1 5 2 15 10

Сравним первые два элемента (выделены красным). Если первый больше второго, то поменяем их местами. В нашем случае это не так (3 меньше 8), поэтому оставим элементы без изменений и сравним второй и третий элементы:

3 8 -1 5 2 15 10

8 больше -1, поэтому произведём обмен. Получим такой массив (обмен выделен синим):

3 -1 8 5 2 15 10

Сравним третий и четвёртый элементы (8 и 5). Поскольку 8 больше 5, то произведём обмен:

3 -1 5 8 2 15 10

Сравним четвёртый и пятый элементы (8 и 2). Снова производится обмен:

3 -1 5 2 8 15 10

Подошла очередь пятого и шестого элементов (8 и 15). В этом случае обмена нет, массив остаётся без изменений:

3 -1 5 2 8 15 10

Последнее сравнение – шестого и седьмого элементов (15 и 10) – снова вызовет обмен. В результате мы получим вот такой массив:

3 -1 5 2 8 10 15

Что мы получили? Благодаря нашим обменам, самое большое число (15) «всплыло» в конец массива, как пузырёк с воздухом всплывает в водоёме. Поэтому данный алгоритм ещё называют **методом пузырька**.

Поскольку самый большой элемент оказался на своём, последнем месте, то в дальнейшем его рассматривать не нужно. Поэтому продолжим производить сравнения с первого элемента до предпоследнего. Посмотрим, какие значения будут принимать элементы массива при следующем проходе массива. (Обмен, как и раньше, будем выделять синим цветом.)

```
-1 3 5 2 8 10 | 15
-1 3 5 2 8 10 | 15
-1 3 2 5 8 10 | 15
-1 3 2 5 8 10 | 15
-1 3 2 5 8 10 | 15
-1 3 2 5 8 | 10 15
```

Сколько раз мы будем повторять проход по массиву? Вы заметили, что у нас всё время увеличивается упорядоченная часть в конце массива? Когда эта упорядоченная часть будет состоять из всех элементов, кроме первого (вспомните, что массив из одного элемента всегда упорядочен), то мы сможем сказать, что массив отсортирован. Это означает, что мы должны сделать $N - 1$ проход, где N – длина массива.

В алгоритме полезно учесть следующее: если, совершая очередной проход по массиву, мы не сделали ни одного обмена, то это означает, что массив уже упорядочен и дальнейшие проходы не требуются.

Составим блок-схему этого алгоритма (рис 1.4).

Обратите внимание на появление логической переменной *flag*. Если значение этой переменной равно *true*, то это означает, что в процессе прохода были обмены, если *false*, то обменов не было. Если происходит обмен ($A[j] \geq A[j+1]$), то переменная *flag* принимает значение *true*. Кроме того, в неё нужно занести значение *true* в самом начале, чтобы мы смогли попасть в цикл.

Посмотрим на условие продолжения внешнего цикла: $(i < n)$ and $flag$. Оно будет ложным в случае, когда либо i станет большим или равным n , либо $flag$ станет равным $false$. То есть цикл закончится тогда, когда либо мы сделаем $n - 1$ проходов, либо в процессе последнего прохода не будет сделано ни одного обмена.

Поскольку после каждого прохода в конце массива образуется упорядоченная часть, то в следующий раз эту часть рассматривать не надо. Для реализации данного условия во вложенном цикле переменная j изменяется до значения $n - i$.

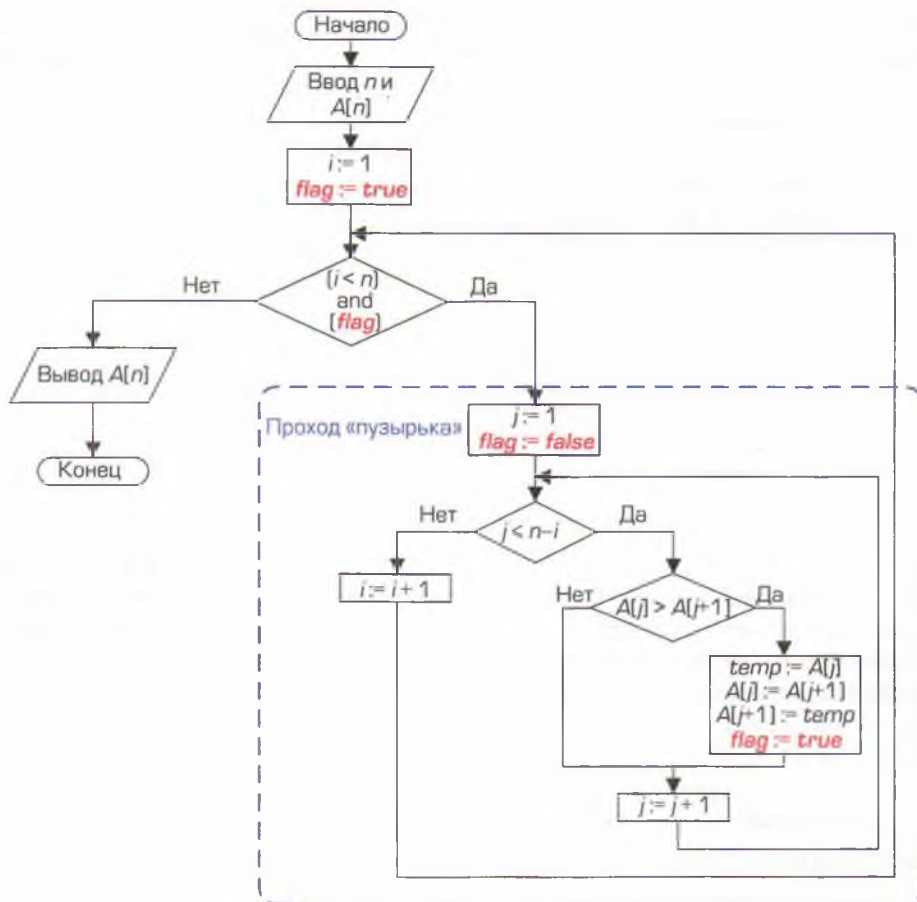


Рис. 1.4

Приведём программу, которая сортирует массив методом пузырька

```
program obmen;
var
  i, j, n, temp: integer;
  flag: boolean;
  A: array [1..100] of integer;
begin
  // Ввод массива
  write('Введите количество элементов в массиве: ');
  readln(n);
  for i := 1 to n do
    begin
      write('A[', i, ']=');
      readln(A[i]);
    end;
  // Конец ввода массива
  // Сортировка пузырьком
  flag := true;
  i := 1;
  while (i < n) and flag do
    begin
      flag := false;
      for j := 1 to n - i do
        if (A[j] >= A[j+1]) then
          begin
            temp := A[j];
            A[j] := A[j+1];
            A[j+1] := temp;
            flag := true;
          end;
      i := i + 1;
    end;
  // Конец сортировки
  // Вывод отсортированного массива
  write('Отсортированный массив: ');
  for i := 1 to n do write(A[i], ' ');
  writeln;
  // Конец вывода массива
end.
```

Отметим, что условие продолжения внешнего цикла

`(i < n) and flag`

можно сократить и записать просто:

```
flag
```

Это возможно, так как на некотором шаге обязательно не будет сделано ни одного обмена и внешний цикл закончится.

Мы рассмотрели два способа сортировки массивов. Оба этих метода используют вложенные циклы для своей работы.

Алгоритм сортировки пузырьком является самым простым в реализации, но в то же время он самый неэффективный. Поэтому применять его можно только на небольших массивах.

Алгоритм выбором эффективен по количеству обменов значений местами и неэффективен по количеству сравнений элементов между собой.

Для упорядочивания больших объёмов данных используются более совершенные методы, но и они в общем случае используют несколько проходов массива.

В наших примерах мы сортировали массивы по неубыванию. Однако критерии упорядочения бывают весьма и весьма разнообразными.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Упорядочение (сортировка) массива – это процесс расстановки элементов массива в порядке, определяемом некоторым правилом. Примеры сортировок: по неубыванию, по невозрастанию.

Существует множество алгоритмов сортировки, как универсальных, так и рассчитанных на конкретные ситуации.

Мы рассмотрели два алгоритма сортировки: выбором и обменом (пузырьком). Первый из них использует в своём составе алгоритм поиска минимального числа.

Сортировка пузырьком самая неэффективная, но в то же время самая простая в реализации.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Можно ли изменить алгоритм сортировки выбором так, чтобы мы искали не минимум, а максимум?

2. Как изменить алгоритм сортировки пузырьком, чтобы массив сортировался по невозрастанию?

3. С помощью блок-схемы на рис. 1.3 реализуйте на языке Паскаль алгоритм сортировки массива выбором.

4. С клавиатуры вводится число n , а за ним массив из n элементов. Проверьте его на упорядоченность по возрастанию.

§ 4. Структурирование программ. Подпрограммы

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Представьте, что две фабрики выпускают однотипный мебельный гарнитур. В гарнитур входят 6 стульев, обеденный стол и комод. Естественно, что каждая фабрика обязана предоставить инструкцию по сборке. У первой фабрики инструкция занимает 8 листов: 6 – на каждый из стульев и по одному для стола и комода. А вторая фабрика выпустила инструкцию только на 3 листах.

- Как, по вашему мнению, второй фабрике удалось так сократить инструкцию? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните свои умения программирования на языке Паскаль (Учебник для 7-го класса, книга 2, модуль 1.)

Как в языке программирования используются циклические конструкции? (Учебник для 7-го класса, книга 2, модуль 1, § 5–6.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7-го класса, книга 2, модуль 1.)

РЕШЕНИЕ ПРОБЛЕМЫ

ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ

Очень часто встречается такая ситуация, когда для решения поставленной задачи приходится использовать ранее созданные алгоритмы.

Представьте, например, что вы проектируете систему, которая предназначена для рисования геометрических фигур. Допустим, вы хотите рисовать квадраты и окружности. Но любые два квадрата отличаются длиной своих сторон, а окружности – размером своих радиусов. Получается, что нужно придумать разные алгоритмы для рисования каждого квадрата и каждой окружности? Конечно, нет. Достаточно создать **вспомогательный алгоритм**, в который вы будете передавать параметр – длину стороны квадрата или радиус окружности. Этот вспомогательный алгоритм сможет строить нужную вам фигуру, а в основном алгоритме вы будете только вызывать вспомогательный с необходимым вам значением параметра.

Для оформления вспомогательных алгоритмов в программировании применяются **подпрограммы**.

ПОДПРОГРАММЫ

В языке Паскаль подпрограммы делятся на **процедуры** и **функции**.

И в процедуру, и в функцию мы можем из основной программы передавать **параметры**.

Функция — это подпрограмма, которая **возвращает** результат — некоторое значение. Например, вы можете написать функцию, в которую будете передавать число, а она будет возвращать квадрат этого числа. Или, например, функцией является подпрограмма, которая вычисляет площадь параллелограмма по заданным длинам его сторон.

- Какие ещё примеры функций вы можете привести?

Процедура — это подпрограмма, которая, в отличие от функции, не возвращает значение. Пример процедуры — подпрограмма, которая по переданному ей числу n выводит n звёздочек.

Подпрограммы бывают встроенными и определёнными программистом. Встроенные подпрограммы являются частью какой-либо компьютерной системы (встроены в неё). Их написали другие программисты, но вы можете ими пользоваться. Оказывается, вы уже неоднократно встречались со встроенными подпрограммами. Например, это процедуры ввода (`read`, `readln`) и вывода (`write`, `writeln`) в Паскале.

Кроме того, язык Паскаль обладает большим количеством встроенных функций. Приведём некоторые из них.

Функция	Описание
<code>sqr (a)</code>	Возводит число a в квадрат
<code>sqrt (a)</code>	Возвращает квадратный корень из a
<code>abs (a)</code>	Возвращает модуль числа a

Теперь вы можете использовать эти функции в своих программах.

А как создавать свои собственные функции и процедуры?

ФУНКЦИИ

В языке Паскаль функции описываются в программе в разделе **var**.

В качестве первого примера разберём структуру встроенной функции `sqr`, написав свою функцию с именем `square`.

```
program f1;
var
  a, s: integer;
function square(x: integer): integer;
begin
  square := x * x;
end;
```

```

begin
  write('Введите число: ');
  readln(a);
  s := square(a);
  writeln('Квадрат числа ', a, ' равен ', s);
end.

```

Как вы видите, структура подпрограммы (в данном случае, функции) очень похожа на структуру самой программы: у неё тоже есть заголовок и содержимое. Давайте обратим внимание на **заголовок функции**:

```
function square(x: integer): integer;
```

Описание функции начинается с ключевого слова **function**, за ним следует имя функции – *square*, в скобках идут **параметры** – данные, которые функция принимает (в нашу функцию передаётся значение типа *integer*), а в конце после символа «:» указывается **тип значения**, которое функция возвращает (здесь мы хотим вернуть значение типа *integer*).

Давайте будем представлять себе функцию как «**чёрный ящик**». Так инженеры и программисты называют объект, об устройстве которого они ничего не знают (но очень хотят узнать), но имеют информацию о том, какие входные параметры объект принимает и какие выходные параметры выдаёт (рис. 1.5).

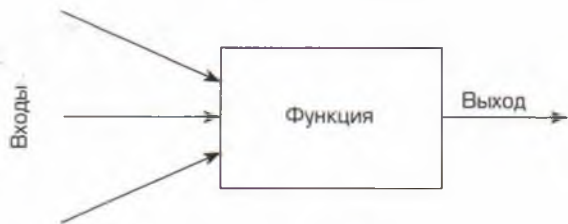


Рис. 1.5

Объявив нашу функцию, мы сообщили компилятору (вспомните, что такое компилятор), что:

- 1) мы хотим создать функцию;
- 2) мы даём ей имя *square*;
- 3) функция будет принимать один целочисленный параметр – *x*;
- 4) функция будет возвращать целое число.

Этими действиями мы создали «чёрный ящик». Теперь мы должны сделать его «прозрачным», то есть написать содержимое (тело) функции:

```

begin
  square := x * x;
end;

```

В строке `square := x * x;` мы говорим следующее: функция *square* возвращает значение квадрата числа *x*.

Функция *square* написана, нам осталось её вызвать, то есть использовать. Делаем мы это в основной программе в строке:

```
s := square(a);
```

Эта строка означает следующее: передать в функцию *square* значение переменной *a*, выполнить функцию и занести в переменную *s* результат её работы.

Обратите внимание, что в описании функции параметр обозначен как *x*, а в её вызове — как *a*. Это означает, что мы передаём функции значение переменной *a*: в функции это будет значение переменной *x*.



Параметры в описании функции называются **формальными** (в нашем примере *x* — формальный параметр), а в обращениях (вызовах) к ней — **фактическими** (в нашем примере *a* — фактический параметр). Фактические параметры — это переменные как раз с теми значениями, которые мы хотим передать функции. В нашей функции значение переменной *a* передаётся соответственно переменной *x*.

Рассмотрим пример более сложной функции — *power*, которая будет возводить вещественное число *a* в степень *n*. Напомним, что $a^n = a \cdot a \cdot \dots \cdot a$ *n* раз. У этой функции уже два параметра, так как мы должны сообщить ей и число, которое нужно возвести в степень, и саму степень. Возвращать эта функция будет вещественное число.

Заголовок функции:

```
function power(base: real; p: integer): real;
```

Нам понадобятся дополнительные переменные *r* и *i*, мы должны описать их в разделе **var** функции.

```
var
  i: integer;
  r: real;
```

Переменные, которые описываются внутри подпрограммы (у нас это переменные *i*, *r*), называются **локальными**. **Локальные переменные** доступны только в подпрограмме, где они описаны. Параметры в подпрограмме тоже являются локальными переменными для этой подпрограммы. В отличие от локальных, **глобальные переменные** описываются в основной программе, их можно использовать и в ней, и во всех подпрограммах. Об использовании глобальных переменных мы поговорим в следующем параграфе.

```

program f2;
var
  n: integer;
  a, res: real;
function power(base: real; p: integer): real;
var
  i: integer;
  r: real;
begin
  r := 1;
  for i := 1 to p do r := r * base;
  power := r;
end;
begin
  write('Введите число a: ');
  readln(a);
  write('Введите степень: ');
  readln(n);
  res := power(a, n);
  writeln('Результат - ', res);
end.

```

Обратите внимание на тот факт, что если в функцию передаются несколько параметров, то в заголовке они перечисляются через символ «;». При вызове функции порядок следования параметров важен. Они должны идти в той же последовательности, в которой соответствующие им параметры описаны в подпрограмме.

В нашем примере в функцию *power* сначала поступает основание степени — *base*, а потом показатель степени — *p*.

```

function power(base: real; p: integer): real;

```

Поэтому и в вызове функции сначала должна идти переменная *a*, содержащая значение, равное основанию степени, а потом — переменная *n*, содержащая показатель степени.

```

res := power(a, n);

```

ПРОЦЕДУРЫ

Процедуры в языке Паскаль описываются в том же месте, что и функции, — в разделе описания переменных **var**. Описания процедур и функций можно чередовать.

Одна подпрограмма может вызывать другую. Программист сам принимает такое решение, руководствуясь только одним правилом: если подпрограмма *A* вызывает подпрограмму *B*, то подпрограмма *B* должна быть описана раньше подпрограммы *A*.

Мы уже говорили о том, что процедура, в отличие от функции, не возвращает значение. Рассмотрим пример процедуры.

```
program p1;
var
  n, k: integer;
procedure print(m: integer);
var
  i: integer;
begin
  for i := 1 to m do write('*');
  writeln;
end;
begin
  write('Введите n: ');
  readln(n);
  for k := 1 to n do print(k);
end.
```

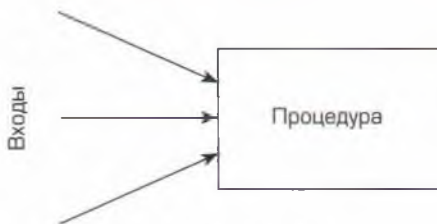


Рис. 1.6

Заголовок процедуры начинается с ключевого слова **procedure**, далее следует её имя, а затем в скобках список параметров. Процедура ничего не возвращает, поэтому указывать тип (как у функции) не нужно.

Можно рассматривать процедуру как «чёрный ящик» (рис. 1.6). В нашем случае, написав следующий заголовок:

```
procedure print(m: integer);
```

мы сообщили компилятору, что:

- 1) мы хотим создать процедуру;
- 2) мы даём ей имя *print*;
- 3) процедура будет принимать один целочисленный параметр – *m*.

Нам понадобилась локальная переменная *i*, с её помощью мы в теле процедуры *m* раз выводим символ «*», после чего переводим курсор на новую строку с помощью вызова процедуры *writeln* без параметров. В этом месте как раз происходит вызов одной процедуры из другой.

§ 5. Передача параметров в подпрограммы

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

На прошлом занятии вы узнали, что такое подпрограммы, и как они устроены. А вы можете привести пример подпрограммы, которая возвращает несколько значений?

✳ Получается? Если не получается, то в чём затруднение? Каких знаний или умений не хватает? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Что такое подпрограммы? (§ 4)

Вспомните всё, что вы узнали о переменных в программировании. (Учебник для 7-го класса, книга 2, модуль 1 и § 1–4 этого учебника.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7-го класса, книга 2, модуль 1.)

РЕШЕНИЕ ПРОБЛЕМЫ

В предыдущем параграфе мы с вами рассмотрели принципы работы с подпрограммами, разобрались, что они собой представляют, и лишь немного поговорили о том, как основная программа передаёт данные в подпрограммы и как получает от них ответ. Давайте рассмотрим правила взаимодействия основной программы и подпрограмм подробнее.

ПЕРЕДАЧА ПАРАМЕТРОВ ПО ЗНАЧЕНИЮ

Когда мы с вами обсуждали подпрограммы, мы представляли их как «чёрные ящики», то есть объекты, в которые из основной программы передаются данные с помощью параметров.

Рассмотрим пример.

Код программы	Вывод
<pre> program param1; var a: integer; procedure p1(b: integer); begin writeln('b=', b); b := b * b; writeln('b=', b); end; begin readln(a); writeln('a=', a); writeln('вызов процедуры'); p1(a); writeln('после вызова процедуры'); writeln('a=', a); end. </pre>	<pre> a=5 вызов процедуры b=5 b=25 после вызова процедуры a=5 </pre>

Мы вводим значение в переменную *a*, выводим его, затем передаём это значение в процедуру *p1*. При вызове процедуры значение *переменной a копируется в переменную b*. Внутри процедуры мы уже работаем только с переменной *b* — изменяем её значение и выводим исходное и новое значения. Мы видим, что значение переменной *a* не меняется во время работы процедуры.

Такой способ передачи параметров называется передачей **по значению**.

ИСПОЛЬЗОВАНИЕ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ

Посмотрите на следующий пример.

Код программы	Вывод
<pre> program param2; var a: integer; procedure p2(b: integer); begin writeln('b=', b); b := b * b; writeln('b=', b); writeln('внутри процедуры'); writeln('a=', a); a := a + 5; end; begin readln(a); writeln('a=', a); writeln('вызов процедуры'); </pre>	<pre> a=5 вызов процедуры b=5 b=25 внутри процедуры a=5 после вызова процедуры a=10 </pre>

Код программы	Вывод
<pre>p2(a); writeln('после вызова процедуры'); writeln('a=', a); end.</pre>	

В этом примере мы выводим значение переменной *a* в процедуре *p2*. Кроме того, внутри процедуры мы *изменяем* значение переменной *a*. Всё это можно сделать, поскольку переменная *a* является **глобальной** по отношению к данной процедуре. Как мы говорили в предыдущем параграфе, глобальные переменные (в данном случае переменная *a*) описываются в основной программе, они доступны и в ней, и в подпрограмме. Обратите внимание, что после выполнения процедуры и возврата в основную программу переменная *a* имеет изменённое значение.

Может получиться так, что программист в качестве имени локальной переменной в подпрограмме будет использовать имя, совпадающее с именем глобальной переменной. В этом случае при обращении к переменной *приоритет будет иметь локальная переменная*.

В следующем примере в процедуре производится работа с локальной переменной *a*, значение глобальной переменной *a* не меняется во время работы процедуры.

Код программы	Вывод
<pre>program param3; var a: integer; procedure p3(b: integer); var a: integer; begin a := 10; writeln('b=', b); b := b * b; writeln('b=', b); writeln('внутри процедуры'); writeln('a=', a); end; begin readln(a); writeln('a=', a); writeln('вызов процедуры'); p1(a); writeln('после процедуры'); writeln('a=', a); end.</pre>	<pre>a=5 вызов процедуры b=5 b=25 внутри процедуры a=10 после процедуры a=5</pre>

ПЕРЕДАЧА ПАРАМЕТРОВ ПО ССЫЛКЕ

Рассмотрим пример.

Код программы	Вывод
<pre> program param4; var a: integer; procedure p4(var b: integer); begin writeln('b=', b); b := b * b; writeln('b=', b); writeln('внутри процедуры'); writeln('a=', a); end; begin readln(a); writeln('a=', a); writeln('вызов процедуры'); p4(a); writeln('после процедуры'); writeln('a=', a); end.</pre>	<pre> a=5 вызов процедуры b=5 b=25 внутри процедуры a=25 после процедуры a=25</pre>

Посмотрите, какая интересная картина получается! Во-первых, у нас появилось слово **var** перед переменной *b* в описании параметров процедуры. Во-вторых, каким-то образом внутри процедуры глобальная переменная *a* получила то же значение, что и локальная *b*. И после завершения работы процедуры переменная *a* своё новое значение сохранила. Что же произошло?

Ключевое слово **var** перед параметром *b* означает, что при вызове процедуры *p4(a)* подпрограмме будет передано не значение переменной *a*, а её *адрес* (ссылка на неё). Подпрограмма будет производить действия с глобальной переменной *a*. Переменная *b* в этом случае как бы прикрепляется, связывается с переменной *a*. Теперь эти переменные неотделимы друг от друга. Если меняется *a*, то меняется и *b*, а если меняется *b*, то своё значение изменит и *a*.

Такой способ передачи параметров называется передачей **по ссылке**.

Фактически, при передаче параметра по значению в подпрограмму передаётся копия переменной, а при передаче по ссылке – сама переменная.

У вас может возникнуть вопрос: зачем нужен такой сложный способ передачи параметров?

Давайте рассмотрим следующую задачу: написать функцию, в которую передаются два целых числа *a* и *b* и которая возвращает *одновременно* остаток от деления *a* на *b* и целую часть этого же деления.

Как же такое сделать, ведь функция возвращает только одно значение? Здесь нам как раз поможет способ передачи параметров по ссылке.

Код программы	Вывод
<pre> program param5; var a, b, div_, mod_: integer; function super(a, b: integer; var d: integer): integer; begin d := a div b; super := a mod b; end; begin writeln('Введите целые числа a и b'); write('a='); readln(a); write('b='); readln(b); mod_ := super(a, b, div_); writeln('a div b = ', div_); writeln('a mod b = ', mod_); End. </pre>	<p>Введите целые числа a и b</p> <p>a=5</p> <p>b=2</p> <p>a div b = 2</p> <p>a mod b = 1</p>

Мы описали четыре глобальные переменные: *a* и *b* – для хранения вводимых чисел, *div_* и *mod_* – для хранения результата. Также мы создали функцию *super* со следующим заголовком:

```
function super(a, b: integer; var d: integer): integer;
```

В эту функцию мы будем передавать наши числа *a* и *b*, а также ещё и параметр *d*, в котором будем получать целую часть от деления *a* на *b*. А функция будет возвращать остаток от деления *a* на *b*.

Вызывается функция *super* следующим образом:

```
mod_ := super(a, b, div_);
```

Таким образом, после вызова функции в переменной *mod_* появится результат (значение) работы функции, а в переменной *div_* – значение, которое приобрела переменная *d* внутри функции, поскольку она передавалась по ссылке. Итак, нам удалось получить два значения: одно – возвращённое функцией, второе – в глобальной переменной. Посмотрите и проанализируйте результат работы программы.

Такой метод очень часто применяется в программировании.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

В языке Паскаль существуют разные способы передавать параметры в подпрограммы.

Первый способ – передача параметров по значению. В подпрограмму передаётся значение глобальной переменной, сама глобальная переменная при этом не изменяется.

Второй способ – использование глобальных переменных. В подпрограммах можно использовать любые глобальные переменные, описанные в основной программе. Замечание: старайтесь использовать этот метод как можно реже.

Третий способ – передача параметров по ссылке. В подпрограмму передаётся адрес глобальной переменной. Параметр связывается с передаваемой глобальной переменной. Работа в подпрограмме производится с глобальной переменной, и она изменяется.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Какие способы передачи параметров в подпрограммы вы узнали?
2. Разберите предлагаемую программу по шагам и ответьте, что будет выведено на экран.

```
program question;  
var  
  a, b, c: integer;  
procedure p(var x, y: integer; z: integer);  
var  
  a: integer;  
begin  
  a := 5; c := 4; y := x + y;  
end;  
procedure q(var x, y, z: integer);  
var  
  a: integer;  
begin  
  a := 5; c := 4; y := x + y;  
end;  
procedure r(x: integer; var y, z: integer);
```

```
var
  a: integer;
begin
  a := 5; c := 4; y := x + y;
end;
begin
  a := 2; b := 5; c := -2; p(a, b, c); writeln('a=', a,
'; b=', b, '; c=', c);
  a := 1; b := 10; c := -2; q(a, b, c); writeln('a=', a,
'; b=', b, '; c=', c);
  a := 1; b := 5; c := -2; r(a, b, c); writeln('a=', a,
'; b=', b, '; c=', c);
end.
```

Проверь себя

Задание 1

Напишите следующую программу на языке Паскаль. Даны целые числа A , B и C . Если $A \leq B \leq C$, то все числа замените их квадратами, если $A > B > C$, то каждое число замените наибольшим из них, в противном случае смените знак каждого числа.

Задание 2

С клавиатуры вводится число n , а за ним — массив из n элементов. Найдите номер последнего по счёту положительного элемента массива.

Задание 3

С клавиатуры вводится число n . Напишите программу с функцией вычисления факториала, которая вычисляет сумму:

$$1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \frac{1}{6!} + \frac{1}{7!} + \dots$$

ПРИМЕНЯЕМ ЗНАНИЯ (необходимый уровень)

Задание 1

Напишите программу, которая строит таблицу истинности для логического выражения $(not\ a) \ or\ (not\ b)$. Решить эту задачу вам помогут примеры, приведённые в § 1.

Задание 2

С клавиатуры вводится число n , за ним – массив из n целых чисел и число x . Найдите количество элементов массива, значения которых равны $x \cdot x$.

Задание 3

Массив целых вводится с клавиатуры до нуля. Ноль при этом не учитывается. Посчитайте количество отрицательных чисел в массиве.

Задание 4

Массив действительных чисел вводится с клавиатуры до нуля, при этом ноль не учитывается. Напишите функцию, которая будет возвращать среднее значение элементов массива. В основной программе выведите это число.
Подсказка: опишите массив как глобальную переменную.

Задание 5

Напишите программу, в состав которой входит процедура *print*, выводящая следующую картинку:

```

*   *   *   *   *   *
*           *
*           *
*           *
*           *
*           *
*   *   *   *   *   *

```

Задание 6

С клавиатуры вводится число n – количество строк в треугольнике. Затем оно передаётся в процедуру *print*, которая должна вывести на экран, например при $n = 4$, следующую картинку:

```

1
1  2
1  2  3
1  2  3  4

```


ПРИМЕНЯЕМ ЗНАНИЯ (повышенный уровень)

Задание 1

Напишите программу, которая строит таблицу истинности для логической формулы $(\text{not } a) \text{ or } (\text{not } b) \text{ or } c$. Решить эту задачу вам помогут примеры, приведенные в § 1. Подумайте, сколько вложенных циклов вам понадобится.

Задание 2

С клавиатуры вводится число n , а за ним — массив из n элементов. Найдите и выведите на экран произведение всех положительных элементов массива с чётными индексами.

Задание 3

Целые числа вводятся до тех пор, пока не будет введён ноль. Верно ли, что все числа равны? *Подсказка:* массив в этой задаче использовать не надо, а вот логические переменные вам помогут.

Задание 4

Напишите программу, содержащую функцию, вычисляющую квадрат натурального числа, используя следующую закономерность:

$$\begin{aligned} 1^2 &= 1; \\ 2^2 &= 1 + 3; \\ 3^2 &= 1 + 3 + 5; \\ 4^2 &= 1 + 3 + 5 + 7. \end{aligned}$$

Задание 5

С клавиатуры вводится натуральное число n , а за ним — массив действительных чисел из n элементов. Напишите функцию, которая вычисляет разность между максимальным и минимальным элементами массива. *Подсказка:* опишите массив как глобальную переменную. Постарайтесь для поиска максимума и минимума использовать один цикл.

Задание 6

С клавиатуры вводится число n — число строк в треугольнике. Затем оно передаётся в процедуру *print*, которая должна вывести на экран, например при $n = 4$, следующую картинку:

```

4
3  4
2  3  4
1  2  3  4

```

ПРИМЕНЯЕМ ЗНАНИЯ (максимальный уровень)

Задание 1

Напишите программу, которая строит таблицу истинности для логического выражения: $(a \text{ or } (\text{not } b)) \text{ and } (c \text{ or } (\text{not } d)) \text{ and } (\text{not } a)$.

Задание 2

С клавиатуры вводится натуральное число, не превышающее 10 000, — номер года. Напишите функцию, которая определяет, является ли данный год високосным. Напомним, что високосными являются года, номера которых кратны 4, но не кратны 100, а также года, номера которых кратны 400. Например, 2000 год — високосный, поскольку 2000 делится на 400, а 1900 год — не високосный, поскольку 1900 делится на 4, но делится и на 100 тоже. Функция должна возвращать значение либо «истина», либо «ложь».

Задание 3

Напишите программу, в которой вводится натуральное число n . Затем это число передаётся в функцию *test*, в которой последовательно с клавиатуры вводятся n натуральных чисел. Функция должна вернуть значение «истина», если все введённые числа являются полными квадратами каких-либо других чисел, и значение «ложь», если это не так. Например, для последовательности 1, 4, 81, 16 должно быть возвращено значение «истина», а для 1, 6, 25 — «ложь».

Задание 4

С клавиатуры вводится число n , а за ним — массив целых чисел из n элементов. Напишите процедуру, которая ищет и выводит два самых больших числа в массиве, при условии, что в массиве есть как минимум два различных элемента. Например, в массиве 1, 3, 9, 5, 7 два самых больших числа — это 9 и 7, а в массиве 1, 3, 3, 3 — это 1 и 3. *Подсказка:* объявите массив, как глобальную переменную.

Задание 5

С клавиатуры вводится число n , а за ним — массив из n чисел. Упорядочите любым способом левую половину массива по возрастанию, а правую — по убыванию. Если в массиве нечётное количество элементов, то центральный элемент отнесите к правой части. Выведите итоговый массив.

Задание 6

С клавиатуры вводится число n . Напишите программу с функцией вычисления факториала, которая вычисляет сумму: $1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \frac{1}{6!} + \frac{1}{7!} + \dots$.

В этой сумме знаки слагаемых чередуются. Вычисление факториала должно быть оформлено отдельной функцией.

Итоговая проверочная работа

Задание 1

а) Что будет выведено на экран в результате работы программы?

```

program d;
var
    a, b: integer;
procedure s1;
var
    a, c: real;
begin
    a := 2.5; b := 5; c := 0;
    write(a, ' ', b, ' ', c);
end;
begin
    a := 2; b := 7;
    write(b, ' ');
    s1;
    writeln(' ', a);
end.

```

б) Напишите функцию *sum* вида **function** sum(n: integer): real, которая вычисляет и возвращает следующую сумму:
 $1 + 1/2 + 1/3 + \dots + 1/n$.

в) С клавиатуры вводятся три числа. Напишите программу, которая проверяет, могут ли эти числа быть длинами сторон прямоугольного треугольника. Выведите «да», если могут, и «нет» в противном случае.

Задание 2

а) Что будет выведено на экран в результате работы программы?

```

program d;
var
    a, c: integer;
procedure s2 (var b: integer);
var
    d, c: real;
begin
    d := a / 5; c := d + 3;
    write(a, ' ');
    b := 5;
    write(d);
end;

```

```

begin
  a := 7; c := a mod 2;
  s2(a);
  writeln(' ', a);
end.

```

- б) Напишите функцию *perfect* вида **function** *perfect*(n: integer): boolean, которая возвращает значение «истина», если число *n* является совершенным, то есть равным сумме всех своих делителей. Например, числа $6 = 1 + 2 + 3$ и $28 = 1 + 2 + 4 + 7 + 14$ – совершенные.
- в) С клавиатуры вводятся действительные числа *a* и *b*. Если *a* и *b* отрицательны, то каждое значение замените его модулем (вам поможет встроенная функция *abs*); если отрицательно только одно из них, то удвойте оба значения; если оба значения неотрицательны и ни одно из них не принадлежит отрезку $[0,5; 2,0]$, то уменьшите оба значения в 10 раз; в остальных случаях оставьте числа без изменения. Выведите результат.

Задание 3

- а) Найдите ошибки в функции и исправьте их.

```

function error (a: real);
var
  z: integer;
begin
  z := a mod 4;
  write(z);
  a := z;
end;

```

- б) Напишите функцию *friend* вида **function** *friend*(a, b: integer): boolean; Функция должна определять, являются ли данные числа дружественными. (Подсказка: Два натуральных числа называются дружественными, если каждое из них равно сумме всех натуральных делителей другого. При этом само число в качестве делителя не рассматривается.)
- в) С клавиатуры вводятся действительные положительные числа *a*, *b*, *c*, *d*. Выясните, можно ли прямоугольник с длинами сторон *a* и *b* уместить внутри прямоугольника с длинами сторон *c* и *d* так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне второго прямоугольника.

§ 6. Знакомство с математической логикой: продолжение

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Вы читали детективные романы, не так ли? Про Шерлока Холмса, Эркюля Пуаро, комиссара Мегрэ? А вы никогда не задумывались, каким же образом этим детективам удавалось распутывать сложные ситуации так же просто и эффектно, как клубок ниток? Им всегда помогал логический способ доказательств их предположений. Представьте, что вам известны некоторые факты. Возможно ли на их основании чётко построить логическое доказательство? Или их мало? Или эти факты противоречат друг другу?

■ Как вы думаете, существуют ли правила решения логических задач? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Что вы уже знаете о математической логике? (§ 1)

РЕШЕНИЕ ПРОБЛЕМЫ

В § 1, посвящённом первому знакомству с математической логикой, мы рассмотрели три основные логические операции: И, ИЛИ, НЕ. С помощью этих операций можно записать любое высказывание. (Вспомните, что такое высказывание.) Однако часто бывает удобнее использовать и другие логические операции. Рассмотрим их.

ОПЕРАЦИЯ «СЛЕДОВАНИЕ»

Рассмотрим высказывание: «Если завтра будет солнце, то я поеду за город». Оно состоит из двух простых высказываний: «Завтра будет солнце» и «Я поеду за город», которые связаны между собой связкой «Если ..., то». Этой связке соответствует логическая операция «**следование**».

Если мы обозначим первое простое высказывание буквой a , а второе – буквой b , то на языке математической логики наше сложное высказывание будет выглядеть так: $a \rightarrow b$ и читаться: «из a следует b ».

Дадим определение операции «следование».

Определение. Значение операции $a \rightarrow b$ – «ложь» в том случае, когда a истинно, а b ложно, и «истина» – в противном случае.

Как записать на языке математической логики следующее высказывание: «Если завтра не будет светить солнце, то Вася не пойдёт гулять?»

1) Обозначим латинскими буквами простые высказывания:

a – «Завтра будет светить солнце»;

b – «Вася пойдёт гулять».

Заметьте, что простым высказыванием является именно «Завтра будет светить солнце», а не «Завтра не будет светить солнце», поскольку второе высказывание – это применение к первому логической операции (НЕ), следовательно, оно уже не простое.

2) Запишем наше сложное высказывание с помощью логических операций: $(\text{НЕ } a) \rightarrow (\text{НЕ } b)$.

Построим таблицу истинности операции «следование».

a	b	$a \rightarrow b$
ложь	ложь	истина
ложь	истина	истина
истина	ложь	ложь
истина	истина	истина

Мы уже сказали, что с помощью логических операций И, ИЛИ, НЕ можно записать любое высказывание. Это означает, что и операцию «следование» можно выразить через них. Давайте попробуем это сделать.

Рассмотрим логическое выражение $(\text{НЕ } a) \text{ ИЛИ } b$ и построим для него таблицу истинности. Сколько у нас в выражении переменных? Две. Значит, строк в нашей таблице будет столько же, сколько и в таблице любой бинарной операции – 4. Но столбцов будет немного больше. Сначала мы вычислим значение операции НЕ a , а потом выполним операцию ИЛИ над значениями в столбцах «НЕ a » и « b ».

a	b	НЕ a	(НЕ a) ИЛИ b
ложь	ложь	истина	истина
ложь	истина	истина	истина
истина	ложь	ложь	ложь
истина	истина	ложь	истина

Сравните получившийся столбец–результат со столбцом–результатом операции «следование». Они совпадают на одинаковых наборах $\{a, b\}$. Таким образом, мы с вами получили важнейший результат: для того чтобы доказать, что логические выражения равны, необходимо построить их таблицы истинности и сравнить итоговые результаты.

Итак, $a \rightarrow b = (\text{НЕ } a) \text{ ИЛИ } b$.

ОПЕРАЦИЯ «ЭКВИВАЛЕНЦИЯ»

Рассмотрим высказывание: «Снег в России выпадает тогда, когда наступит зима». Это высказывание состоит из двух простых высказываний: «Снег в России выпадает» и «Наступает зима», а также связки «тогда, когда» (связка может формулироваться ещё и так: «тогда и только тогда, когда»).

Логическая операция, соответствующая этой связке, называется «**эквиваленция**». На языке математической логики она обозначается так: $a \sim b$ и читается: « a эквивалентно b ».

Определение. Значение операции $a \sim b$ – «истина» в том случае, когда a и b имеют одинаковые значения, и «ложь» – в противном случае.

Построим таблицу истинности для операции «эквиваленция».

a	b	$a \sim b$
ложь	ложь	истина
ложь	истина	ложь
истина	ложь	ложь
истина	истина	истина

Выразим операцию «эквиваленция» через операции И, ИЛИ, НЕ:

$$a \sim b = (a \text{ И } b) \text{ ИЛИ } ((\text{НЕ } a) \text{ ИЛИ } (\text{НЕ } b)).$$

ОПЕРАЦИЯ «ИСКЛЮЧАЮЩЕЕ ИЛИ»

Рассмотрим высказывание: «Я съем либо апельсин, либо яблоко». В отличие от операции ИЛИ, в случае применения которой оба исхода могут случиться одновременно, в данном случае может произойти либо первый вариант, либо второй, но не два одновременно.

Логическая операция, соответствующая связке «либо ..., либо» («или..., или»), называется «**исключающее ИЛИ**» («строгое ИЛИ») и на языке математической логики обозначается \oplus .

Определение. Значение операции $a \oplus b$ – «истина» в том случае, когда a и b имеют разные значения, и «ложь» – в противном случае.

Построим таблицу истинности для операции «исключающее ИЛИ».

a	b	$a \oplus b$
ложь	ложь	ложь
ложь	истина	истина
истина	ложь	истина
истина	истина	ложь

Выразим операцию «исключающее ИЛИ» через операции И, ИЛИ, НЕ:

$$a \oplus b = (a \text{ И } (\text{НЕ } b)) \text{ ИЛИ } ((\text{НЕ } a) \text{ И } b).$$

ОБОЗНАЧЕНИЯ ОСНОВНЫХ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Обратите внимание, что для обозначения рассмотренных в этом параграфе операций мы применяли специальные значки. Для основных операций И, ИЛИ, НЕ тоже существуют обозначения:

Операция	Обозначение
И	$a \& b$
ИЛИ	$a \vee b$
НЕ	\overline{a}

Примечание. Операцию ИЛИ обозначают также «+».

Тогда зависимости рассмотренных в параграфе логических операций от основных операций будут выглядеть следующим образом:

$$a \rightarrow b = \overline{a} \vee b$$

$$a \sim b = a \& b \vee \overline{a} \& \overline{b}$$

$$a \oplus b = a \& \overline{b} \vee b \& \overline{a}$$

ПОИГРАЕМ В ШЕРЛОКА ХОЛМСА

Давайте немного поиграем в Шерлока Холмса и попробуем применить математическую логику для решения логических задач.

Задача 1. На вопрос сыщика, кто из троих задержанных преступников взломал сейф, был получен ответ: «Если взломал первый, то взламывал и второй, но неверно, что если взломал третий, то взламывал и второй». Кто из преступников взломал сейф?

Любые логические задачи надо решать не спеша и последовательно. Сначала выделим из нашего исходного высказывания два поменьше:

- 1) «Если сейф взломал первый, то взламывал и второй»;
- 2) «Неверно, что если сейф взламывал третий, то взламывал и второй».

Теперь обозначим буквами простые высказывания, из которых состоят эти два сложных:

- 1) a – «Сейф взломал первый преступник»;
- 2) b – «Сейф взломал второй преступник»;
- 3) c – «Сейф взломал третий преступник».

Давайте запишем на языке математической логики известные нам высказывания.

1) Если сейф взломал первый, то взламывал и второй: $a \rightarrow b$.

2) Неверно, что если сейф взламывал третий, то взламывал и второй: $\overline{c \rightarrow b}$.

Второе высказывание состоит из отрицания к операции «следование».

Нам известно, что оба наших высказывания выполняются одновременно.

Это значит, что если мы применим к этим высказываниям операцию И, то её результат должен быть истинным. Поэтому нам остаётся проверить, при каких же значениях a , b и c значение выражения $(a \rightarrow b) \& c \rightarrow b$ будет равно «истина». Для этого построим таблицу истинности для указанного логического выражения. В этой таблице истинности будет 8 строк, поскольку у нас 3 переменные. (Вспомните, сколько будет строк в таблице истинности для логического выражения, содержащего 2 переменные.) Для краткости будем обозначать в таблице значения «истина» и «ложь» как «И» и «Л» [рис. 1.8].

a	b	c	$a \rightarrow b$	$c \rightarrow b$	$\overline{c \rightarrow b}$	$(a \rightarrow b) \& \overline{c \rightarrow b}$
Л	Л	Л	И	И	Л	Л
Л	Л	И	И	Л	И	И
Л	И	Л	И	И	Л	Л
Л	И	И	И	И	Л	Л
И	Л	Л	Л	И	Л	Л
И	Л	И	Л	Л	И	Л
И	И	Л	И	И	Л	Л
И	И	И	И	И	Л	Л

Рис. 1.8

Как вы видите, итоговое логическое выражение принимает значение «истина» только в одной строке (в таблице она выделена синим цветом). Посмотрим, какая из наших переменных истинна в этом случае. Это переменная c . Поэтому сыщик будет абсолютно точно уверен, что сейф взломал третий преступник и остальных надо пока отпустить.

Задача 2. По обвинению в ограблении банка перед судом предстали Джон, Билл и Марк. Следствием установлено, что:

- 1) если Джон не виновен или Билл виновен, то Марк виновен;
- 2) если Джон не виновен, то Марк не виновен.

Судья никак не мог рассудить это дело и обратился к Холмсу. Тот подумал немного и сказал: «Могу сказать точно: Джон виновен!». Как же он это сделал?

Обозначим буквами простые высказывания, из которых состоят сложные:

- 1) d – «Джон виновен»;
- 2) b – «Билл виновен»;
- 3) m – «Марк виновен».

Запишем на языке математической логики данные задачи:

- 1) $(\bar{d} \vee b) \rightarrow m$;
- 2) $\bar{d} \rightarrow \bar{m}$.

Рассмотрим операцию И над нашими условиями:

$$[(\bar{d} \vee b) \rightarrow m] \& (\bar{d} \rightarrow \bar{m}).$$

Построим для полученного логического выражения таблицу истинности и обратим внимание на строки, где его значение истинно (рис. 1.9).

d	b	m	\bar{d}	\bar{m}	$\bar{d} + b$	$(\bar{d} + b) \rightarrow m$	$\bar{d} \rightarrow \bar{m}$	$[(\bar{d} + b) \rightarrow m] \& (\bar{d} \rightarrow \bar{m})$
л	л	л	и	и	и	л	и	л
л	л	и	и	л	и	и	л	л
л	и	л	и	и	и	л	и	л
л	и	и	и	л	и	и	л	л
и	л	л	л	и	л	и	и	и
и	л	и	л	л	л	и	и	и
и	и	л	л	и	и	л	и	л
и	и	и	л	л	и	л	и	л

Рис. 1.9

В обеих строках, где значение итогового выражения истинно, переменная d принимает значение «истина», переменная b – значение «ложь», а m принимает оба значения. Это означает, что Джон точно виновен, а Билл невиновен. А вот про Марка, к сожалению, ничего точно сказать нельзя. Для этого нам не хватает данных.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

В § 1 вы изучили три основные логические операции: И, ИЛИ, НЕ. В этом параграфе вы познакомились с логическими операциями «следование», «эквиваленция», «исключающее ИЛИ».

Все логические операции можно выразить через три основные.

Для решения логических задач необходимо:

1. Обозначить буквами все простые высказывания, из которых состоят сложные, приведённые в условии.
2. Записать на языке математической логики сложные высказывания.
3. Применить операцию И ко всем высказываниям из условия.
4. Построить для полученной формулы таблицу истинности и проанализировать результат.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Перечислите логические операции, которые вы знаете.
2. Докажите правильность равенств, выражающих операции «исключающее ИЛИ» и «эквиваленция» через три основные логические операции.
3. Решите ещё одну логическую задачу про Шерлока Холмса.

Вернувшись домой, Холмс позвонил доктору Ватсону.

– Говорит Холмс. Есть новости?

– Да. Поступили сообщения от инспекторов Скотленд-Ярда:

1) Джонс установил, что если Смит был пьян, то или Джек убийца, или Смит лжёт.

2) Гаррисон считает, что или Джек убийца, или Смит не был пьян и убийство произошло после полуночи.

3) Инспектор Лейстрейд просил передать Вам, что если убийство произошло после полуночи, то или Джек убийца, или Смит лжёт.

4) Затем звонил...

– Всё. Спасибо. Этого достаточно.

Шерлок Холмс положил трубку. Он знал, что трезвый Смит никогда не лжёт. Теперь он знал всё.

Что же знал Холмс?

§ 7. Использование констант и собственных типов

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Скажите, вам больше нравятся удобные или неудобные вещи? Возьмем, например, обувь. Согласитесь, что в удобной обуви ходить гораздо приятнее. Вопрос удобства можно рассматривать и по отношению к программам. Они могут быть написаны так, что любая работа с ними программиста может быть удобной, может быть не очень удобной, а может быть совсем неудобной. Заметьте, что мы всё время говорим о хорошем стиле программирования.

- Какая важная проблема здесь поставлена? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните свои умения программирования на языке Паскаль.

Что понимается под хорошим стилем программирования? (Учебник для 7-го класса, книга 2, модуль 1, § 2.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7-го класса, книга 2, модуль 1.)

Что такое отладка программ и как она выполняется? (Учебник для 7-го класса, книга 2, модуль 1, § 7.)

РЕШЕНИЕ ПРОБЛЕМЫ

КОНСТАНТЫ

Константа — это такой объект, который никогда не меняет своего значения. Само слово «константа» происходит от латинского *constanta*, что означает «постоянная, неизменная». Где же в программировании применяются константы?

Представьте себе такую ситуацию, что вы пишете программу, которая анализирует температуру за окном, постоянно сравнивает её с эталоном и сообщает об отклонениях. Допустим, вам сказали, что эталон — это 20°C, и вы написали программу, в тексте которой много раз присутствует сравнение значения текущей температуры с 20.

И тут в момент, когда вы уже закончили свою работу, заказчик вам говорит: «Теперь у нас эталон — 22°C. И вам не остаётся ничего иного, как исправлять везде числа с 20 на 22. А если таких исправлений сотни?

Неудобно, правда?

Но не расстраивайтесь! Из этой ситуации есть выход.

Мы можем описать **константу**, у неё есть имя и значение. Константа очень похожа на переменную, её можно использовать в операциях языка. Единственное ограничение заключается в том, что константу изменять нельзя.

Константы описываются в разделе **const**, который располагается после ключевого слова **program**. Этот раздел не является обязательным – может отсутствовать в программах. Начинается он с ключевого слова **const**. Затем следуют описания констант в формате:

```
<Имя_константы> = <Значение_константы>
```

```
const
```

```
    MAX = 5;
```

```
    MIN = 2;
```

```
    Etalon = 20;
```

Рассмотрим фрагмент программы анализа температуры, использующей константы.

```
program c1;
```

```
const
```

```
    etalon = 20;
```

```
begin
```

```
    for i := 1 to n do
```

```
        if A[i] <> etalon then writeln('Внимание!
                                   Несоответствие эталону!')
```

```
end.
```

Обратите внимание на то, как константы похожи на переменные.

Что мы теперь будем делать в ситуации, когда заказчик попросит нас изменить значение эталона? Мы просто поменяем значение константы *etalon* в разделе **const**.

Очень часто константы используются в описаниях массивов для указания индексных границ. Выглядит это следующим образом:

```
const
```

```
    MAX = 100;
```

```
var
```

```
    a: array [1..MAX] of integer;
```

Такой способ описания массивов относится к *хорошему стилю программирования*.

КАК ПЕРЕДАТЬ В ПОДПРОГРАММУ МАССИВ?

Давайте попробуем передать в подпрограмму массив таким же образом, как мы передавали в неё переменные.

```
procedure test(a: array [1..100] of integer);  
begin
```

```
    ***  
end;
```

При запуске такой программы вы получите ошибку:

Program1.pas(4): Тип параметра или возвращаемого значения не может быть описанием записи или описанием массива с границами

Дело в том, что для передачи массивов в подпрограмму мы должны назвать тип массива по-другому, одним словом. Это делается в специальном разделе программы, который называется **type**.

РАЗДЕЛ TYPE

В разделе **type** находятся описания **собственных типов**, то есть типов, которые создает сам программист.

Раздел **type** располагается сразу после раздела **const**, если он есть в программе, а при отсутствии описания констант — после ключевого слова **program**. Этот раздел необязателен, как и раздел **const**, но при использовании массивов в программах *хорошим стилем* является его наличие.

Начинается раздел ключевым словом **type**.

Затем следуют описания типов в формате:

```
<имя_типа> = <описание_типа>
```

После такого описания вновь созданный тип можно использовать наравне со встроенными типами языка, такими как *integer* и *real*.

Рассмотрим пример программы.

```
program t1;  
type  
    my_array = array [1..100] of integer;  
var  
    a: my_array;  
    i, n: integer;  
procedure print(b: my_array; n: integer);  
var  
    i: integer;
```

```
begin
  for i := 1 to n do write(b[i], ' ');
  writeln;
end;
begin
  write('Введите количество элементов в массиве: ');
  readln(n);
  for i := 1 to n do
    begin
      write('a[', i, ']=');
      readln(a[i]);
    end;
  print(a, n);
end.
```

В разделе **type** мы описали новый тип, который назвали *my_array*. Правила именования типов такие же, как и правила именования переменных. (Вспомните их.) Тем самым мы сообщили программе, что теперь под типом *my_array* нужно понимать массив из 100 целых чисел.

- Проанализируйте приведённую программу. Что она делает?

Теперь мы можем использовать это имя в любых местах программы, где описываются переменные, в том числе и в описании параметров подпрограмм.

Массивы можно передавать в подпрограммы и по ссылке. Принцип остаётся таким же, что и для обычных переменных. При изменении массива, описанного в подпрограмме, будет изменяться и глобальный массив.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Для упрощения модернизации программ, а также для того, чтобы программы легко читались, применяются константы. Константы описываются в разделе **const**, который располагается после ключевого слова **program**. Использование констант не является обязательным.

Работа с константами очень похожа на работу с переменными. Отличие заключается в том, что константу нельзя изменять.

Использование констант в описании массивов относится к хорошему стилю программирования.

Программист может создавать свои собственные типы, описания которых размещаются в разделе **type**, который находится после раздела **const**. Использование собственных типов не является обязательным, кроме случая передачи массивов в подпрограммы.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Зачем нужны константы в программе на языке Паскаль?
2. Посмотрите на фрагмент программы. Что делает процедура *p*?

```
program p1;  
const  
    MAX = 100;  
type  
    my_array = array [1..MAX] of integer;  
var  
    A: my_array;  
    l: integer;  
procedure p(var M: my_array; n: integer);  
var  
    i: integer;  
begin  
    for i := 1 to n do  
        if (M[i] mod 2 <> 0) then M[i] := 0;  
    end;
```

3. Напишите программу на языке Паскаль с функцией *f*, в которую передается массив вещественных чисел и которая вычисляет сумму значений элементов этого массива.

§ 8. Работа с упорядоченными массивами

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Представьте, что вы пришли в библиотеку и перед вами находится большой алфавитный каталог. Этот каталог можно представить в виде массива, элементы которого упорядочены по алфавиту. Но даже если бы это было не так, вы уже умеете упорядочивать массивы и смогли бы исправить эту оплошность. И вы знаете, что в упорядоченном массиве гораздо эффективнее искать нужные данные.

• А почему эффективнее? Как вы думаете, какие именно алгоритмы поиска позволяет выполнять упорядоченный массив? Много ли их? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните свои навыки работы с массивами. (§ 2–3 и учебник для 7–го класса, книга 2, модуль 1, § 8–9.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7–го класса, книга 2, модуль 1.)

РЕШЕНИЕ ПРОБЛЕМЫ

ЭФФЕКТИВНЫЙ АЛГОРИТМ ПОИСКА

Мы уже говорили, что упорядочивание массивов даёт нам большие преимущества при работе с данными. Давайте посмотрим, как осуществить поиск в отсортированном по неубыванию массиве.

Итак, представьте, что перед вами на столе лежат N пронумерованных от 1 до N шариков, причём номера расположены по неубыванию. И ваш друг загадал число от 1 до N . Ваша задача – найти тот шарик, номер которого загадал ваш друг, или сказать, что такого шарика у вас нет. Как же это сделать?

Для определённости положим, что шарики пронумерованы от 1 до 20. Поделите общее количество шариков пополам и посмотрите на «центральный» шарик. Он будет иметь номер 10. Спросите вашего друга: «Ты загадал номер 10?». Если он ответит положительно, то это означает, что вы нашли то, что искали. Если ответ отрицательный, то спросите: «Загаданный номер больше 10?». Если он ответит «да», то вы будете искать нужный шарик справа от центрального, если «нет» (номер меньше 10), то слева.

Этот принцип применим только в случае упорядоченного массива, поскольку если загаданное число больше центрального, то находится оно может

только справа от него. Ведь все элементы, расположенные левее центра, заведомо его меньше.

Допустим, ваш друг сказал, что загаданный шарик имеет номер больше 10, поэтому вам нужно рассматривать часть массива от 11 до 20. Опять посмотрите на центральный шарик. Он будет иметь номер 15. Снова задайте вопрос другу: «Я нашёл твоё загаданное число?». «Нет, — ответит друг, — оно меньше 15».

Не беда. Значит, это число находится левее 15. Рассмотрите часть массива от 11 до 14. Возьмите центральный шарик — с номером 12. «Да! — ответит ваш друг. — Ты угадал!».

Описанный способ поиска называется **двоичным (бинарным)**, поскольку мы всё время делим массив на две части. Причём, эти две части могут быть неравными. Например, ничего не мешает нам разделить массив не пополам, а в отношении один к трём. От способа деления зависит только скорость поиска. В нашем алгоритме мы будем придерживаться деления пополам, и делать это будем так: сложим номер левой границы и номер правой границы и разделим сумму нацело на два.

$$\begin{aligned}(1 + 20) \div 2 &= 10, \\ (11 + 20) \div 2 &= 15, \\ (11 + 14) \div 2 &= 12.\end{aligned}$$

Сравните получившиеся числа с номерами центральных шаров из примера.

Когда мы понимаем, что нам надо рассмотреть начальную («левую») часть массива, то мы объявляем в качестве новой правой границы уменьшенный на 1 номер центрального элемента, в случае перехода к «правой» части массива новой левой границей становится увеличенный на 1 номер центрального элемента (центр — 10, следовательно, новая левая граница — 11; центр — 15, следовательно, новая правая граница — 14).

Алгоритм останавливается, если на некотором шаге центральный элемент оказывается равным искомому или в какой-то момент левая граница становится больше правой. В таком случае искомого элемента в массиве нет.

Рассмотрим ещё один пример. Пусть задан массив: 1, 3, 5, 6, 8, 10, 15, 25. Он упорядочен по неубыванию. Необходимо найти индекс элемента со значением 5. Алгоритм поиска будет следующим:

1. В массиве 8 элементов, поэтому изначально левая граница будет равна 1, а правая — 8.
2. Анализируем центральный элемент. Его номер = $(1 + 8) \div 2 = 4$. Проверим, равен ли он 5. Нет, не равен: его значение — 6. Поскольку $5 < 6$, то дальше будем искать слева от него.
3. Левая граница останется прежней — 1, а правая изменится и станет на 1 меньше, чем значение центрального элемента, то есть $4 - 1 = 3$. Левая граница не превысила правую, поэтому мы можем продолжать поиск.

4. Снова анализируем центральный элемент. Его номер = $(1 + 3) \div 2 = 2$. Его значение – 3. $5 > 3$, следовательно, будем искать в правой части.
 5. Теперь изменяется левая граница. Она будет на единицу больше 2, то есть станет равной 3. Правая же останется без изменения – 3. Левая граница пока не превысила правую. Продолжаем поиск.
 6. Ищем центр – $(3 + 3) \div 2 = 3$. Третий элемент массива имеет значение 5. Ура! Мы нашли искомый элемент. Он имеет индекс 3.
- Нарисуем блок-схему этого алгоритма (рис. 1.10).

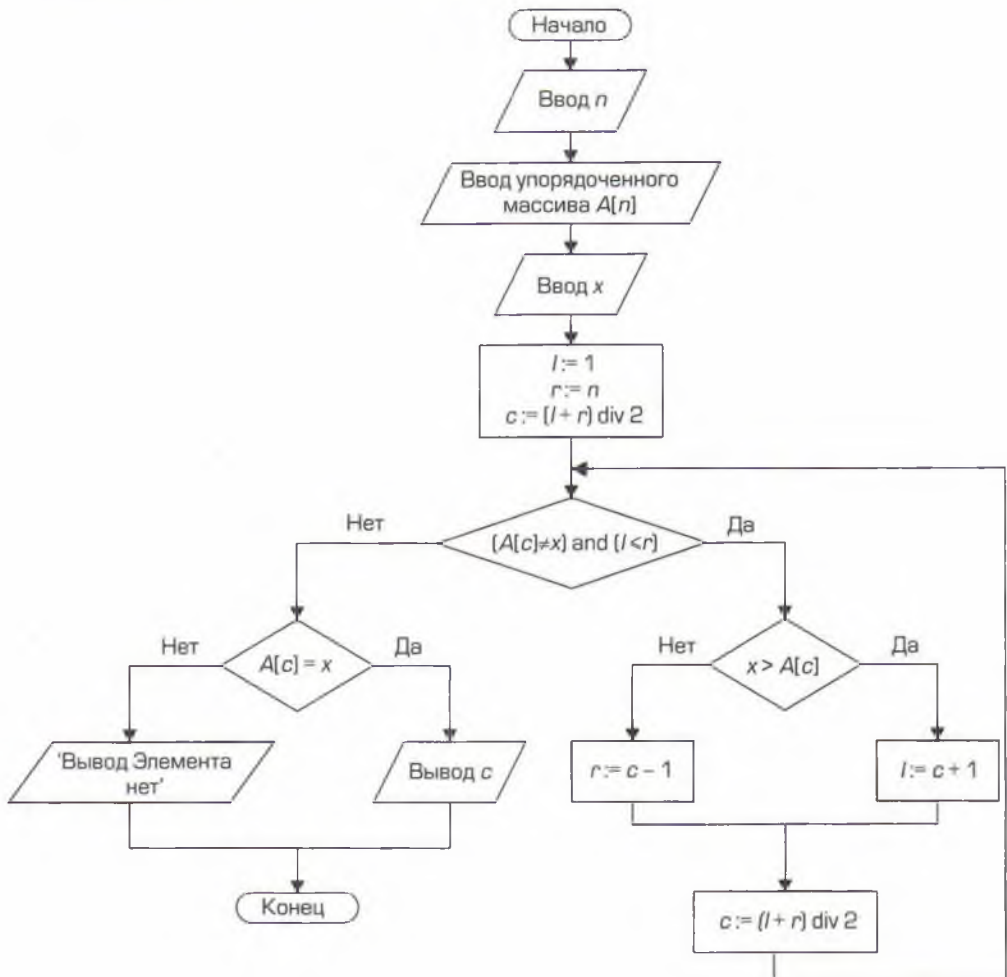


Рис. 1.10

В данной блок-схеме переменная l хранит левую границу, а переменная r – правую. Выход из основного цикла $[A[c] \neq x] \text{ and } (l \leq r)$ возможен или в случае, когда мы нашли элемент (ложно первое логическое выражение), или когда левая граница превысила правую (ложно второе логическое выражение).

Итак, мы рассмотрели эффективный алгоритм поиска элемента в упорядоченном массиве. Этот алгоритм не производит полного перебора, как это делает линейный поиск, однако всё же существуют случаи, когда линейный поиск работает быстрее.

- Как вы думаете, в каких ситуациях линейный поиск работает быстрее, чем бинарный?

Эффективность поиска оценивают *в среднем*. Это означает, что берут множество вариантов заполнения массивов, множество искомых чисел и в каждом случае измеряют время работы алгоритма. Затем берут среднее значение всех вычисленных значений времени, и это значение считается средним значением эффективности алгоритма. После этого остаётся сравнить средние значения различных методов и сделать вывод.

В наших примерах мы рассматривали массивы, упорядоченные по неубыванию. Однако сортировка может происходить по любому критерию. Алгоритм в этом случае, конечно, претерпит изменение, но суть его (деление на части и дальнейший поиск в одной из частей) при этом не меняется.

СЛИЯНИЕ МАССИВОВ

Очень часто встречаются задачи, в которых нужно объединить уже имеющиеся данные из нескольких массивов в один итоговый. Такие задачи называются слиянием данных. В случае неупорядоченных массивов у нас нет никаких вариантов для манёвров. Необходимо добавить элементы второго массива в конец первого и отсортировать результат.

Рассмотрим слияние двух упорядоченных массивов.

Пусть у нас есть два упорядоченных массива: $A = \{1, 3, 4, 8, 9, 10\}$ и $B = \{2, 4, 6, 7\}$. Как вы видите, они упорядочены по неубыванию. Нам нужно сформировать массив C , в котором окажутся все элементы массивов A и B и который будет также упорядочен по неубыванию.

Будем хранить в переменной $m1$ индекс текущего элемента первого массива, в переменной $m2$ – индекс текущего элемента второго массива, а в переменной $m3$ – индекс текущего элемента третьего, формируемого, массива. В начале и $m1$, и $m2$, и $m3$ равны 1.

Сравним $A[m1]$ и $B[m2]$. Если $A[m1] < B[m2]$, то в $C[m3]$ мы занесём $A[m1]$ и $m1$ увеличим на 1, в противном случае в $C[m3]$ мы занесём $B[m2]$ и $m2$ увеличим на 1. Увеличим $m3$ на 1. В нашем примере в первый элемент массива C запишется число 1.

Будем повторять такое сравнение и запись до тех пор, пока один из массивов не закончится.

При этом в массив будет C последовательно заполняться так:

1

1, 2

1, 2, 3

1, 2, 3, 4

1, 2, 3, 4, 4

1, 2, 3, 4, 4, 6

1, 2, 3, 4, 4, 6, 7

Оставшиеся же элементы из другого массива просто добавим в конец массива C .

В результате массив C будет выглядеть так: {1, 2, 3, 4, 4, 6, 7, 8, 9, 10}.

Главное преимущество этого алгоритма заключается в том, что заполнение массива C происходит *за один проход*. Здесь нет вложенных циклов, которые присутствуют в алгоритмах сортировки. Подобное преимущество возможно только потому, что массивы, которые мы объединяем в один, уже являются упорядоченными. Поэтому данный алгоритм выполняется очень быстро.

Рассмотрим его блок-схему (рис. 1.11).

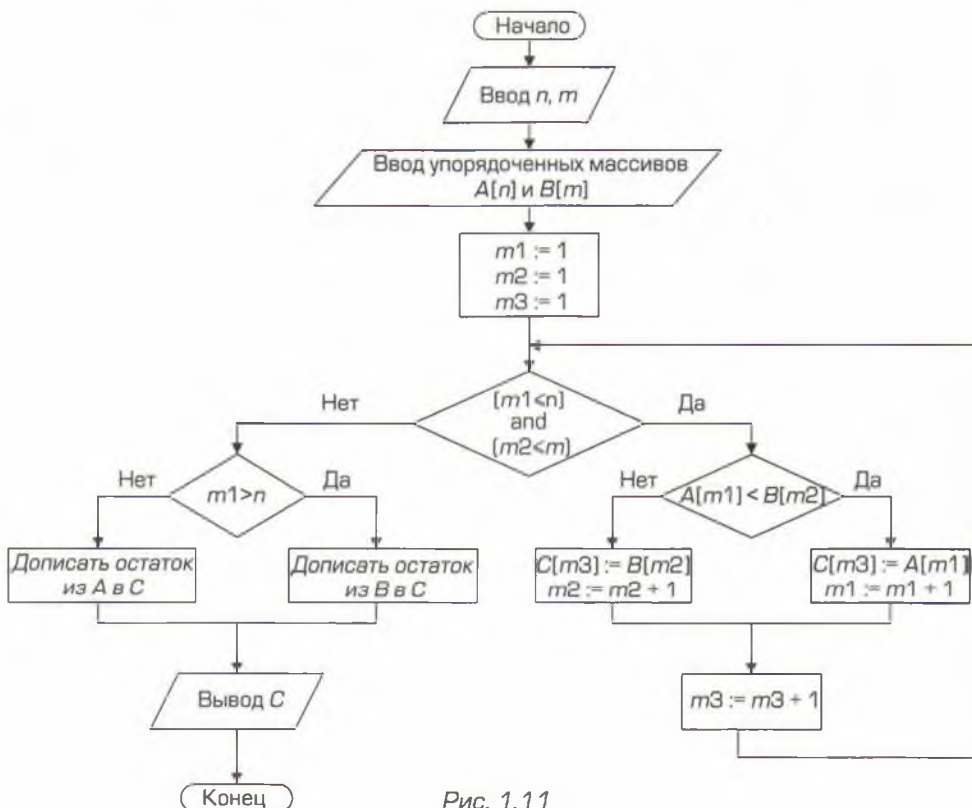


Рис. 1.11

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Когда массивы данных отсортированы (упорядочены), то алгоритмы поиска и работы с информацией становятся гораздо эффективнее.

Бинарный поиск позволяет быстрее, чем линейный, находить элементы в массиве за счёт того, что он делит массив на две части и продолжает при необходимости поиск только в одной из них, в зависимости от критерия поиска.

Алгоритм слияния двух массивов позволяет без использования вложенных циклов объединить в один массив данные из двух упорядоченных массивов.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Какие преимущества имеет поиск в упорядоченном массиве по отношению к поиску в неупорядоченном?
2. Что мы подразумеваем, когда говорим про один алгоритм, что он в среднем эффективнее другого?
3. Как надо изменить блок-схему алгоритма бинарного поиска, если известно, что массив упорядочен по невозрастанию?
4. Напишите на языке Паскаль программу, содержащую функцию, которая реализует двоичный поиск элемента в массиве. Функция должна принимать 3 параметра: количество элементов в массиве, сам массив и искомый элемент. Возвращать функция должна номер элемента, если этот элемент в массиве присутствует, или 0, если элемента в массиве нет. Ввод и вывод реализуйте в основной программе.

§ 9–10. Поговорим об эффективности

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Ребята на уроке создавали программу, которая помогала рассчитать, как перевезти груз, состоящий из множества коробок, с помощью одной грузовой машины. Когда учитель посмотрел на их решения, то он сказал: «Ваше решение нельзя назвать неверным. Но если бы мне нужно было перевозить вещи, то я обратился бы в другую транспортную компанию. Ваш вариант переезда был бы для меня слишком дорогим».

✳ Как вы думаете, почему у учителя могло сформироваться такое мнение? Сформулируйте основной вопрос урока. Сравните свой вариант с авторским (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Какова структура программы в языке Паскаль?

Как в языке программирования используются циклические конструкции? (§ 3 и учебник для 7–го класса, книга 2, модуль 1, § 5–6.)

Вспомните свой опыт работы в интегрированной среде разработки программ. (Учебник для 7–го класса, книга 2, модуль 1.)

Вспомните свои навыки работы с массивами. (§ 2, 3, 8 и учебник для 7–го класса, книга 2, модуль 1, § 8–9.)

РЕШЕНИЕ ПРОБЛЕМЫ

ЭФФЕКТИВНОСТЬ АЛГОРИТМОВ

Когда мы начинали разговор о программировании (в 7–м классе), мы говорили о том, что алгоритмов решения одной и той же задачи может быть много. Программист должен выбрать тот способ решения задачи, который в данном случае наиболее *эффективен*.

Понятие эффективности сложное. Часто случается так, что самое простое решение задачи – то, которое первым приходит в голову, бывает самым неэффективным. Задание программисту на разработку алгоритма включает **критерии эффективности**. Они могут быть такими:

- 1) программа должна работать быстро;
- 2) программа должна использовать определённые алгоритмы;
- 3) программа должна занимать мало ресурсов (памяти или процессора);
- 4) программа должна содержать определённые синтаксические конструкции и т. д.

При этом часто один критерий противоречит другому.

Выбор критерия эффективности алгоритма сродни выбору условий поездки. Представьте, что вы хотите поехать из Москвы в Санкт-Петербург. И тут возникают вопросы:

- 1) Какое время вы хотите потратить на поездку? Если вам надо попасть в город на Неве за 4 часа, то надо ехать на поезде «Сапсан» или лететь на самолёте.
- 2) Если вам абсолютно неважно время, затраченное на путь, но вы хотите потратить минимум средств, то ваш выбор однозначен – ночной поезд и плацкартное купе.
- 3) А если вы едете втроем или вчетвером, то вам выгоден автомобиль.

И таких вариантов может быть очень много. Выбирайте.

ПОИСК ПРОСТЫХ ЧИСЕЛ

В качестве примера поиска эффективного алгоритма рассмотрим следующую задачу: вводится натуральное число $a \geq 2$. Необходимо вывести все простые числа на интервале от 1 до a .

Как вы думаете, какое первое решение приходит в голову? Вот такое:

1. Начало.
2. Положить x равным 2.
3. Проверить, является ли x простым числом.
4. Если x – простое число, то вывести его.
5. Увеличить x на 1.
6. Если x стало больше, чем a , то закончить выполнение программы (перейти на шаг № 7), иначе перейти на шаг № 3.
7. Конец.

А как проверить число x на простоту? Будем проверять его делимость на все числа из интервала $[2; x - 1]$. Если оно разделится хотя бы на какое-то из чисел данного интервала, значит, оно составное, а если не разделится, то простое.

Вот как будет выглядеть эта программа:

```
program simple1;  
var  
    a, x, y: integer;  
begin  
    write('Введите натуральное число a >= 2: ');  
    readln(a);  
    x := 2;  
    while (x <= a) do
```

```

begin
  y := 2;
  // Пока x не делится на y,
  // будем увеличивать y
  while (x mod y <> 0) do y := y + 1;
  // Если x = y, то число простое,
  // и мы его выводим
  if (x = y) then write(x, ' ');
  x := x + 1;
end;
writeln;
end.

```

Вывод программы:

Введите натуральное число $a \geq 2$: 100

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
73 79 83 89 97

Программа получилась очень компактной, но давайте задумаемся, эффективна ли она? Возьмём, например, число 4. Мы выяснили, что оно составное, поскольку делится на 2. Значит, все числа, которые делятся на 4 (8, 12, 16 и так далее) заведомо составные. А мы их заново проверяем.

Получается, что наша программа неэффективна. Но, конечно, только с точки зрения (критерия) быстроты работы. С точки зрения лаконичности кода она нас устраивает. Как же нам улучшить программу? Воспользуемся алгоритмом, который называется **решетом Эратосфена**:

1. Начало.
2. Выписать подряд все целые числа от 2 до a (2, 3, 4, ..., a).
3. Занести в x значение 2 — первое простое число.
4. Вычеркнуть из списка все числа, делящиеся на x , то есть числа $2x$, $3x$, $4x$ и т. д. На первом шаге это будут все чётные числа.
5. Найти первое незачёркнутое число, большее x , и занести его в x . В первый раз это будет число 3.
6. Повторить шаги № 4 и 5 до тех пор, пока x не станет больше, чем $a \div 2$.
7. Все оставшиеся невычеркнутыми числа в списке — простые. Вывести их.
8. Конец.

Приведём текст программы, реализующий этот алгоритм.

```

program simple2;
var
  x, a, y: integer;
  M: array [2..200] of integer;

```


begin

```

write('Введите натуральное число a >= 2: ');
readln(a);
// заполним массив единицами, то есть
// предположим, что все числа - простые
for x := 2 to a do M[x] := 1;
// начинаем с x = 2
for x := 2 to (a div 2) do
    if M[x] <> 0 then
        begin
            // 'Зачёркиваем' все числа, кратные x,
            // то есть заносим 0 в массив на кратные x места
            y := 2 * x;
            while (y <= a) do
                begin
                    M[y] := 0;
                    y := y + x;
                end;
            end;
        // выводим из итогового массива
        // только индексы, которым
        // соответствуют ненулевые элементы
        for x := 2 to a do
            if (M[x] <> 0) then write(x, ' ');
        writeln;
end.
```

Как вы видите, в этой программе нам пришлось использовать массив, в котором индекс равен числу, а значение элемента с этим индексом говорит о том, простое это число или составное. Если i -й элемент массива равен 1, то число простое, а если он равен нулю, то число составное.

Изначально мы занесли в массив все единицы, предполагая, что все числа простые.

Далее мы перебираем числа x , начиная от 2 и заканчивая $a \text{ div } 2$.

- Подумайте, почему нас устраивает этот диапазон и не надо перебирать числа до a .

Если текущее значение $M[x]$ не равно нулю (то есть если x — простое), мы будем «зачёркивать числа», то есть присваивать ноль всем элементам, индексы которых кратны x .

В результате мы выведем все индексы отличных от нуля элементов массива. Это и будут простые числа.

Мы хотим ещё раз обратить ваше внимание на тот факт, что программа с точки зрения объёма кода получилась больше, чем предыдущая, но с точки зрения эффективности по времени она на порядок лучше.

ВЫЧИСЛЕНИЕ ФАКТОРИАЛОВ

Рассмотрим задачу вычисления суммы $S[n] = \frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \dots + \frac{n}{n!}$.

Для её решения мы должны каждый раз вычислять факториал нового числа.

Но можно ли этого не делать? Давайте посмотрим, как соотносятся числа, стоящие в знаменателях дробей этой суммы. Сначала мы должны посчитать $1!$, затем $2!$, затем $3!$ и т. д. А как связан $3!$ с $2!$? Вот так:

$3! = 2! \cdot 3$, $4! = 3! \cdot 4$ и т. д.

$n! = n \cdot (n - 1)!$

Таким образом, зная $(n - 1)!$, вычислить $n!$ просто. Надо только умножить известное значение на n . Но чтобы начать вычисление последовательности факториалов, нужно знать значение $1!$. А оно нам известно.

В результате эффективная программа, вычисляющая заданную сумму, будет выглядеть так:

```
program summa;
var
  n, i, fact: integer;
  s: real;
begin
  // Ввод n
  write('Введите n: ');
  readln(n);
  // Изначально 1! = 1, а сумма = 0
  // Занесём эти значения в fact и s
  fact := 1;
  s := 0;
  // Считаем сумму
  for i := 1 to n do
    begin
      fact := fact * i;
      s := s + i / fact;
    end;
  writeln('Сумма равна: ', s);
end.
```

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Для решения задачи могут быть найдены различные алгоритмы. Выбор конкретного алгоритма определяется критерием эффективности. Хороший программист всегда должен искать эффективное решение своей задачи.

РЕШЕНИЕ ПРОБЛЕМЫ

ВЫЧИСЛЕНИЕ НАИБОЛЬШЕГО ОБЩЕГО ДЕЛИТЕЛЯ

Вычисление наибольшего общего делителя (НОД) двух чисел – задача, которая очень часто встречается, например, при решении задач, связанных с шифрованием. Напомним, что НОД двух чисел a и b (обозначается $\text{НОД}(a, b)$) – это наибольшее число, на которое делятся оба числа. Обычно вычисление НОД реализуется функцией, в которую поступают два числа a и b и которая возвращает их НОД.

Понятно, что если оба числа не равны нулю, то их НОД не может превышать меньшее из них. Поэтому мы можем составить такой алгоритм:

1. Начало.
2. Если какое-либо из чисел a и b равно нулю, то НОД – это другое из них. Закончить выполнение программы (перейти на пункт № 6).
3. Положить $m = \min[a, b]$.
4. Проверить делимость a на m и b на m . Если оба числа делятся на m , то m и есть $\text{НОД}(a, b)$. Если хотя бы одно из чисел не делится нацело на m , то уменьшить m на 1.
5. Если $m \neq 1$, то перейти на пункт № 4. В противном случае $\text{НОД}(a, b) = 1$.
6. Конец.

Вроде бы всё хорошо. Мы получили простой алгоритм. Однако взгляните в него. Мы всё время вычитаем из числа m единицу. А если это число очень большое, например порядка 10^{10} ? В этом случае алгоритм будет работать очень долго, даже на современных мощных компьютерах.

Выходом является использование современной модификации **алгоритма Евклида**.

Пусть у нас есть два целых неотрицательных числа a и b , $a < b$. Тогда a можно представить в виде: $a = b \cdot q + k$. Допустим, число c является $\text{НОД}(a, b)$. Это означает, что и a делится на c , и b делится на c . Следовательно, k тоже делится на c . Поэтому мы можем записать: $\text{НОД}(a, b) = \text{НОД}(b, k)$. Таким образом, мы

уменьшили значения чисел, НОД которых мы ищем, ведь k – это остаток от деления a на b , а он заведомо меньше b .

У какой пары чисел мы всегда знаем НОД? $\text{НОД}(a, 0) = a$.

Последовательная замена чисел приведёт к тому, что в конце концов второе число обратится в 0. Таким образом, первое число из пары и будет НОД.

Общий алгоритм будет выглядеть так:

$$\text{НОД}(a, b) = \begin{cases} a, & \text{при } b = 0; \\ \text{НОД}(b, a \bmod b), & \text{при } b \neq 0. \end{cases}$$

Рассмотрим несколько примеров вычисления НОД по алгоритму Евклида.

$$1) \text{НОД}(5, 0) = 0;$$

$$2) \text{НОД}(24, 16) = \text{НОД}(16, 8) = \text{НОД}(8, 0) = 8;$$

$$3) \text{НОД}(9, 12) = \text{НОД}(12, 9) = \text{НОД}(9, 3) = \text{НОД}(3, 0) = 3;$$

$$4) \text{НОД}(11, 7) = \text{НОД}(7, 4) = \text{НОД}(4, 3) = \text{НОД}(3, 1) = \text{НОД}(1, 0) = 1.$$

Обратите внимание на третий пример. В нём первое число меньше второго. Чтобы можно было применить алгоритм Евклида, потребовалось поменять их местами. Это сделать можно, так как $\text{НОД}(a, b) = \text{НОД}(b, a)$. При $a < b$ алгоритм будет работать неверно. Давайте разделим нацело 9 на 12: $9 = 0 \cdot 12 + 9$. Целая часть от деления 9 на 12 – это 0, а остаток – 9.

В последнем примере НОД двух чисел стал равным единице. Такие числа называются *взаимно простыми*.

Приведём текст функций, одна из которых реализует первый рассмотренный алгоритм вычисления НОД, а вторая – алгоритм Евклида.

```
function simple_nod(a, b: integer): integer;
var
  m: integer;
begin
  if (a * b = 0) then m := a + b
  else
    begin
      if (a > b) then m := b else m := a;
      while (m > 1) and ((a mod m <> 0) or (b mod m <> 0))
        do m := m - 1;
      end;
      simple_nod := m;
    end;
end;
function evklid_nod(a, b: integer): integer;
var
  m: integer;
```

```

begin
  while (b < > 0) do
    begin
      m := b;
      b := a mod b;
      a := m;
    end;
    evklid_nod := a;
  end;

```

Когда a и b одновременно равны нулю, их НОД не определён. Функции выдают в этом случае 0.

БЫСТРОЕ УМНОЖЕНИЕ ЦЕЛЫХ ЧИСЕЛ

Алгоритм, о котором мы сейчас хотим рассказать, используется для того, чтобы «научить» компьютер умножать. Согласитесь, звучит странно, но давайте разбираться! Что означает умножение a на b ? Это означает, что надо сложить a само с собой b раз. Но что делать, если числа a и b большие? Простой алгоритм теряет свою эффективность. Значит, надо придумать что-то иное, более эффективное.

Давайте рассмотрим метод умножения двух целых неотрицательных чисел, который хоть и называется «русским», но был известен ещё в Древнем Египте.

Возьмём два числа: $a = 53$, $b = 36$. Будем составлять таблицу: в первом столбце будем хранить преобразованное число a , во втором — преобразованное число b , в третьем ставить флажок, если b нечётное (рис. 1.12).

Правила преобразования следующие: на каждом шаге мы будем a умножать на 2, а b целочисленно делить на 2.

a	b	Флажок
53	36	
106	18	
212	9	✓
424	4	
848	2	
1696	1	✓

Рис. 1.12

После заполнения таблицы сложим те значения a , для которых соответствующее значение b нечётно: $S = 212 + 1696 = 1908$.

Приведём блок–схему этого алгоритма (рис. 1.13).

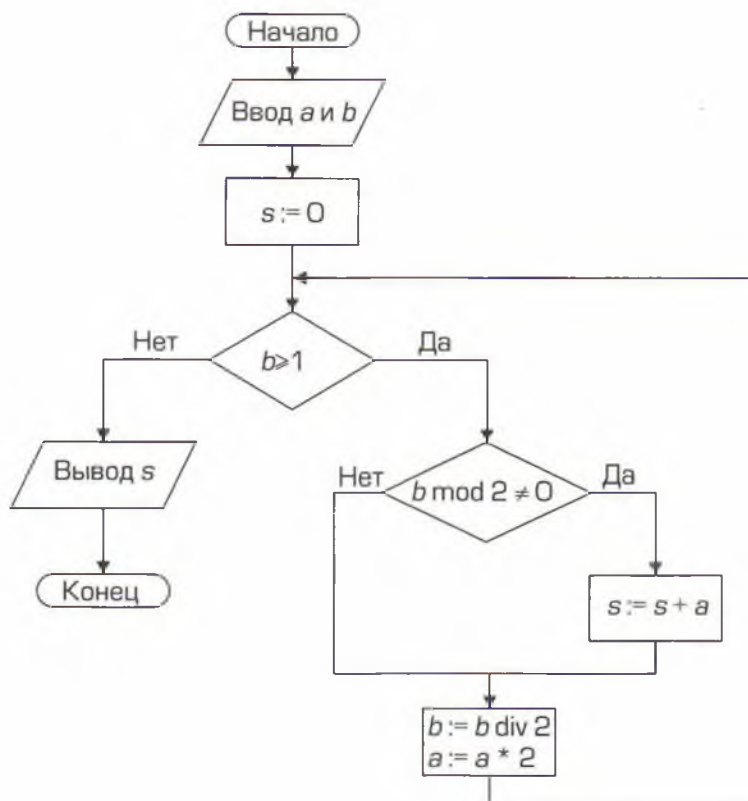


Рис. 1.13

В приведённом алгоритме мы используем умножение на 2 и деление на 2? Но операции умножения на 2 и деления на 2 выполняются процессором очень быстро, так же как сложение и вычитание, это связано с тем, что любые данные в компьютере хранятся в двоичной системе счисления. Но о системах счисления мы с вами поговорим в следующем модуле.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Алгоритм нахождения НОД двух целых чисел часто применяется в шифровании.

Самые быстрые операции с вещественными числами – сложение, вычитание, умножение на 2 и деление на 2.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Почему поиск простых чисел методом перебора неэффективен?
2. Напишите программу на языке Паскаль, в состав которой входит функция, реализующая «русский» метод умножения целых чисел.

Проверь себя

Задание 1

Выразите через основные логические операции логическое выражение: $a \rightarrow b$.

Задание 2

Напишите функцию, в которую поступает массив A и количество элементов в нём. Функция должна вернуть количество положительных элементов массива.

Задание 3

Вариант алгоритма Евклида для натуральных чисел выглядит так:

$$\text{НОД}(a, b) = \begin{cases} a, & \text{при } a = b; \\ \text{НОД}(a - b, b), & \text{при } a > b; \\ \text{НОД}(a, b - a), & \text{при } b > a. \end{cases}$$

Напишите функцию, реализующую этот алгоритм.

ПРИМЕНЯЕМ ЗНАНИЯ (необходимый уровень)

Задание 1

Напишите программу, которая выводит таблицу истинности для логического выражения. Вспомните, как логические операции из этого выражения представляются через основные логические операции. Сделайте вывод красивым. Вспомните также, каким образом форматируется вывод целых и логических переменных в языке Паскаль.

Задание 2

С клавиатуры вводится натуральное число N . Узнайте, сколько натуральных делителей имеет это число. Помните об эффективности и попробуйте понять, какой диапазон чисел нужно проверять.

Задание 3

С клавиатуры вводятся натуральные числа a и b . Напишите функцию, которая вычисляет их **наименьшее общее кратное** (НОК). НОК можно вычислить по формуле:

$$\text{НОК}(a, b) = \frac{a \cdot b}{\text{НОД}(a, b)}$$

Задание 4

Напишите на языке Паскаль программу, в которой есть процедура, обеспечивающая слияние двух упорядоченных по неубыванию массивов. Сами массивы и количество элементов в них опишите, как глобальные переменные. Ввод и вывод реализуйте в основной программе. *Подсказка:* воспользуйтесь блок-схемой из § 8.

Задание 5

С клавиатуры вводятся натуральные числа a и b ($a < b \leq 100$). Найдите и выведите все простые числа на отрезке $[a; b]$.

ПРИМЕНЯЕМ ЗНАНИЯ (повышенный уровень)

Задание 1

Решите логическую задачу.

В нарушении технологического процесса по изготовлению микросхем подозреваются четыре технолога – Алексеев, Бантиков, Воропаев и Гудков. Известно, что:

- 1) если Алексеев нарушил процесс, то и Бантиков тоже нарушил;
- 2) если Бантиков нарушил, то Воропаев нарушил или Алексеев не нарушил;
- 3) если Гудков не нарушил, то Алексеев нарушил, а Воропаев не нарушил;
- 4) если Гудков нарушил, то и Алексеев нарушил.

Кто из подозреваемых нарушил правила?

Для решения этой задачи напишите программу, которая построит таблицу истинности для логического выражения, получившегося в процессе решения. *Подсказка:* в этой задаче должны быть 4 логических переменных.

Задание 2

С клавиатуры вводится натуральное число N . Заполните массив всеми натуральными делителями этого числа в порядке убывания. Само число в качестве делителя учитывать не надо. Выведите массив. Заполнение массива организуйте в отдельной процедуре, вывод – по вашему усмотрению. Помните об эффективности решения.

Задание 3

С клавиатуры вводятся целое число a и натуральное число b . Сократите дробь a/b и, если возможно, выделите целую часть. Например, если $a = -5$ и $b = 10$, то в качестве ответа надо вывести $-1/2$. Если $a = 10$ и $b = 8$, то вывод должен быть: $1\ 1/4$.

Задание 4

Напишите программу на языке Паскаль, в которой есть эффективная функция *summa*, имеющая заголовок

```
function summa (n: integer): real;
```

вычисляющая следующее выражение:

$$S = \frac{2}{1!} - \frac{2^2}{2!} + \frac{2^3}{3!} - \frac{2^4}{4!} + \dots + (-1)^{n+1} \cdot \frac{2^n}{n!}.$$

Ввод n и вывод ответа необходимо реализовать в основной программе. Под вывод выделите 10 знаков, из них 4 – под дробную часть.

Задание 5

С клавиатуры вводятся натуральные числа a и b . Напишите функцию, которая определяет, являются ли они близнецами (функция должна возвращать тип *boolean*). Близнецами называются простые числа, отличающиеся на 2. Близнецами являются, например, числа 5 и 7, 11 и 13, 17 и 19. *Подсказка:* в этой программе нужно написать две функции: одну основную проверочную и вторую, вспомогательную, проверяющую число на простоту.

ПРИМЕНЯЕМ ЗНАНИЯ (максимальный уровень)

Задание 1

Решите логическую задачу.

Наступила осень. Школа собиралась на туристический слёт. Но встал вопрос: какая же будет погода на выходные? Ребята стали искать прогноз в Интернете. Но, к сожалению, все сайты говорили загадками. Вот что удалось выведать:

- 1) «Если не будет ветра, то будет пасмурно и без дождя»;
- 2) «Если будет пасмурно, то обязательно пройдёт дождь, но ветра совсем не будет»;
- 3) «Если будет дождь, то будет пасмурно и без ветра».

Помогите ребятам определить погоду.

Для решения задачи напишите программу, которая построит таблицу истинности для необходимого логического выражения. *Подсказка:* вспомните про то, как выражаются разные логические операции через основные.

Задание 2

С клавиатуры вводятся два натуральных числа a и b . Напишите функцию, которая определяет, являются ли эти числа дружественными. Два натуральных числа называются дружественными, если каждое из них равно сумме всех натуральных делителей другого (само число в качестве делителя не рассматривается).

Задание 3

С клавиатуры вводится натуральное число n , а за ним n натуральных чисел. Найдите их НОД. Постарайтесь не использовать в этой задаче массивы.

Подсказка: $\text{НОД}(a, b, c) = \text{НОД}(\text{НОД}(a, b), c)$.

Задание 4

Известно, что:

$$A_0 = 1;$$

$$A_k = k \cdot A_{k-1} + \frac{1}{k}, \text{ где } k = 1, 2, \dots$$

С клавиатуры вводится натуральное число n . Напишите процедуру, которая вычисляет числа A_0, \dots, A_n и заполняет ими массив. Ввод n и вывод массива организуйте в основной программе. Массив должен быть передан в функцию в качестве параметра.

Задание 5

С клавиатуры вводится натуральное число n . Напишите процедуру, которая раскладывает его на простые множители и выводит это разложение. Ответ надо вывести в виде $n = x * y * \dots * z$. Например, $12 = 2 * 2 * 3$, $15 = 3 * 5$, $100 = 2 * 2 * 5 * 5$, $7 = 7$. *Подсказка:* для решения этой задачи посмотрите внимательно на приведённые в параграфе примеры.

Итоговая проверочная работа

Задание 1

- а) Докажите, что $a \rightarrow b = \bar{a} + b$.
- б) Напишите процедуру, в которую поступает натуральное число n и которая выводит на экран все натуральные делители числа n , меньшие его.

Задание 2

- а) Докажите, что $a \oplus b \oplus b = a$.
- б) Напишите функцию, в которую передаётся число n и которая вычисляет выражение: $(2 + \frac{1}{1}) \cdot (2 + \frac{1}{2}) \cdot (2 + \frac{1}{3}) \cdot \dots$

Задание 3

- а) Постройте таблицу истинности для выражения $a \& (a \rightarrow b) \& (a \rightarrow \bar{b})$ и докажите, что оно всегда ложно.
- б) Напишите процедуру, в которую поступает массив A , число элементов в нём n и число x . Необходимо вернуть массив, из которого надо удалить все вхождения числа x . *Подсказка:*
 - а) можно использовать вспомогательный массив;
 - б) подумайте, каким способом надо передать параметры в процедуру;
 - в) учтите, что после удаления количество элементов в массиве может измениться.



Модуль 2. Системы счисления

Этот модуль поможет вам:

- понять и узнать ещё кое-что о компьютере и о числах;
- изучать числа в позиционных системах счисления, узнать, как записывается число в общем виде в любой системе счисления, как производятся умножение и деление в любой системе счисления и вычисления не только с целыми числами, но и с дробями;
- понять, зачем нужны двоичная, восьмеричная, шестнадцатеричная системы счисления;
- узнать, как представлены в памяти компьютера целые и дробные (вещественные) числа, как осуществляется операция сложения в ячейках памяти.

Для этого вам надо научиться:

- получать целое число, следующее за данным целым числом в любой позиционной системе счисления;
- переводить целые числа в любую систему счисления;
- складывать и вычитать в любых системах счисления;
- переводить в произвольную систему счисления правильную десятичную дробь;
- умножать и делить числа, записанные в любой системе счисления;
- разбираться, как происходит сложение двоичных чисел в ячейках памяти.

Введение



Дорогие друзья!

Вы уже много знаете о компьютере, пользуетесь его помощью для написания рефератов, обработки фотографий, слушаете музыку, общаетесь с друзьями. Нелегко перечислить всё, что заставляет вас обратиться к нему. Но как компьютер работает со всей перечисленной информацией?

Любая информация, вводимая в компьютер, сначала должна быть представлена в виде, доступном для обработки компьютером, — закодирована. В этом модуле мы будем говорить о кодировании чисел. Вы уже знакомы с одним из способов кодирования — десятичной системой счисления. Кроме десятичной системы счисления, существуют и другие способы представления чисел. В этом разделе вы познакомитесь с различными системами счисления, в том числе с двоичной системой счисления, в которой кодируются числа в компьютере.

§ 1. Что такое системы счисления

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Вам уже встречались различные способы записи чисел: пункты плана в сочинении, главы в книге часто нумеруются посредством римских чисел, а для вычислений вы пользуетесь десятичными числами. Но вы знаете, что X – это 10, а I – это 1, и уже привыкли к этому. Вы знаете, что римское число III ($I + I + I$) отличается по значению от десятичного числа 111 ($1 \cdot 100 + 1 \cdot 10 + 1$), хотя в записи каждого из них используются три единицы. А десятичное число 20 и римское число XX оба означают два десятка, а устроены по-разному.

- Что здесь для вас новое, а о чём вы хотите спросить? Сформулируйте основной вопрос урока. Сравните свою формулировку с авторской (с. 141 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните, что вы знаете о разных способах записи чисел.

РЕШЕНИЕ ПРОБЛЕМЫ

- Прочитайте рассказ о различных способах записи чисел и попробуйте объяснить, что такое система счисления.

Люди научились считать предметы очень давно и придумали способы записи результатов счёта – в виде чисел.

Самым древним способом записи чисел была **унарная система** (от латинского слова *unus* – единственный). В ней для записи чисел использовался всего один какой-нибудь знак. Количество предметов обозначалось соответствующим количеством одинаковых знаков (например, число 20 можно было изобразить с помощью двадцати одинаковых подряд идущих вертикальных «палочек»).

В **славянской системе** нумерации для записи чисел использовались все буквы алфавита (правда, с нарушением алфавитного порядка), над которыми ставился знак ~ (титло). Эти знаки являлись цифрами славянской системы. Каждая буква-цифра обозначала количество единиц, десятков или сотен:

$\tilde{а}$ (аз) – 1; $\tilde{в}$ (веди) – 2; $\tilde{г}$ (глаголь) – 3; $\tilde{д}$ (добро) – 4;
 $\tilde{е}$ (есть) – 5; $\tilde{с}$ (зело) – 6; $\tilde{з}$ (земля) – 7; $\tilde{и}$ (иже) – 8; $\tilde{ѡ}$ (фита) – 9;
 $\tilde{і}$ (и) – 10; $\tilde{к}$ (како) – 20; $\tilde{л}$ (люди) – 30 и т. д.

В **римской системе** записи чисел используется следующий набор знаков-цифр, обозначающих числа: I – 1; V – 5; X – 10; L – 50; C – 100; D – 500; M – 1000.

Все другие числа строятся как комбинации цифр в соответствии со следующими правилами:

- если цифра с меньшим значением стоит справа от большей цифры, то их значения суммируются; если слева, то меньшее значение вычитается из большего;
- несколько подряд идущих одинаковых цифр суммируются;
- цифры I, X, C и M могут следовать подряд не более трёх раз каждая;
- цифры V, L и D могут использоваться в записи числа не более одного раза.

Рассмотренные способы (системы) записи чисел называются системами счисления.

Под **системой счисления** понимается способ представления чисел с помощью некоторого алфавита знаков – **цифр**.

Существуют позиционные и непозиционные системы счисления.

В **непозиционных системах счисления** значение цифры в числе не зависит от её позиции в числе.

Например, в числе XXX (30) каждая из цифр означает 10 единиц, независимо от позиции цифры. Поэтому римская система счисления является непозиционной. Непозиционные системы счисления были и у греков и египтян.

Непозиционные системы счисления обладали рядом недостатков. Для записи чисел нужно было использовать много знаков. В непозиционных системах трудно было производить арифметические действия.

По мере развития науки у людей возникла потребность в большем количестве вычислений, что подтолкнуло древних математиков к созданию более удобных способов записи чисел. Появились системы счисления, в которых значение цифры в числе зависело от её позиции в числе. Они были придуманы почти одновременно в Вавилоне, Индии, у древних майя. Привычная нам позиционная десятичная система счисления зародилась не позднее V века нашей эры в Индии и через арабские страны пришла в Европу.

- Вы догадались, почему такие системы счисления называются позиционными? Назовите отличие позиционной системы счисления от непозиционной.

В **позиционной системе счисления** значение цифры в числе определяется её позицией (**разрядом**) в числе.

Например, запись числа $N = 222$ в десятичной системе счисления означает, что в первой позиции (единицы) – 2 единицы, во второй позиции (десятки) – 2 десятка, то есть $2 \cdot 10 = 20$ единиц, в третьей позиции (сотни) – 3 сотни, то есть $2 \cdot 100 = 200$ единиц.

Важной характеристикой системы счисления является её основание.

Основание позиционной системы счисления – это количество цифр в алфавите этой системы.

Основание обычно обозначается латинской буквой « p ».

Например, десятичная система счисления – позиционная, с основанием 10 ($p=10$). Цифры, принятые для записи чисел (алфавит), – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

У систем счисления с основаниями, меньшими 10, в качестве знаков-цифр используются привычные нам цифры десятичной системы, например, в пятеричной системе цифрами являются 0, 1, 2, 3, 4. В двоичной системе счисления это цифры 0, 1. Двоичная система счисления используется для представления чисел в компьютере.

Значение основания приписывают в виде нижнего индекса после последней цифры числа: 25_{10} , 432_5 , 101_2 и т. д.

Число в десятичной системе счисления можно представить в виде суммы, где каждое слагаемое – это цифра числа, умноженная на основание ($p=10$) в степени, определяемой разрядом этой цифры. Например, десятичное число $123,67_{10}$ можно представить в виде:

$$123,675_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 6 \cdot 10^{-1} + 7 \cdot 10^{-2} + 5 \cdot 10^{-3}.$$

Точно так же можно записать число в любой позиционной системе счисления. Например:

$$23,01_5 = 2 \cdot 5^1 + 3 \cdot 5^0 + 0 \cdot 5^{-1} + 1 \cdot 5^{-2}.$$

Такая запись называется **развёрнутой формой записи числа**.

Пример 1

$$N = 371,33_{10}$$

Степени основания:	2	1	0		-1	-2
Цифры числа:	3	7	1	,	3	3

$$N = 3 \cdot 10^2 + 7 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 3 \cdot 10^{-2}$$

Пример 2

$$N = 371,33_8$$

Степени основания:	2	1	0		-1	-2
Цифры числа:	3	7	1	,	3	3

$$N = 3 \cdot 8^2 + 7 \cdot 8^1 + 1 \cdot 8^0 + 3 \cdot 8^{-1} + 3 \cdot 8^{-2}$$

- Какое основание у этой системы счисления? Какие цифры используются для записи чисел в этой системе счисления?

Последовательность чисел, составленная из представленных в развёрнутой форме сомножителей p^i , где p – основание системы счисления, i – степень основания, определяемая разрядом, представляет собой **базис** системы счисления. Базис является второй характеристикой позиционной системы счисления. Базис задаёт значение цифры числа в соответствующем разряде, как ещё говорят, – вес разряда.

Базисы некоторых систем счисления

Степень основания	Основание			
	10	2	3	8
$-k$	10^{-k}	2^{-k}	3^{-k}	8^{-k}
П...
-3	$10^{-3}=0,001$	$2^{-3}=1/8$	$3^{-3}=1/27$	$8^{-3}=1/512$
-2	$10^{-2}=0,01$	$2^{-2}=1/4$	$3^{-2}=1/9$	$8^{-2}=1/64$
-1	$10^{-1}=0,1$	$2^{-1}=1/2$	$3^{-1}=1/3$	$8^{-1}=1/8$
0	$10^0=1$	$2^0=1$	$3^0=1$	$8^0=1$
1	$10^1=10$	$2^1=2$	$3^1=3$	$8^1=8$
2	$10^2=100$	$2^2=4$	$3^2=9$	$8^2=64$
3	$10^3=1000$	$2^3=8$	$3^3=27$	$8^3=512$
...
n	10^n	2^n	3^n	8^n

Но что делать, если основание системы счисления больше 10? Например, широко используется шестнадцатеричная система счисления – её основание равно 16. Значит, и цифр в её алфавите должно быть 16! Выход нашёлся: первые 10 взяли из десятичной системы счисления: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а остальные шесть стали обозначать буквами латинского алфавита: A, B, C, D, E, F.

Все позиционные системы счисления «устроены» одинаково. Правила, по которым получаются числа в позиционных системах, тоже одинаковые для всех позиционных систем счисления.

Давайте разберёмся, как получается целое число, следующее за данным числом.

- Расскажите, как это происходит в десятичной системе счисления.

Конечно, вы правы. Чтобы получить следующее целое число, надо к предыдущему прибавить 1. Но не забудьте, что мы ещё не научились складывать числа в произвольной системе счисления. Поэтому воспользуемся следующим правилом.

Чтобы получить число, следующее за данным, нужно заменить крайнюю правую (младшую) цифру этого числа на следующую за ней в алфавите

системы счисления, причём старшая цифра в алфавите меняется на 0. Если цифра числа стала равной нулю, то следует заменить вторую справа цифру на следующую и т. д. Если нулём стала крайняя правая (старшая) цифра числа, то нужно приписать к числу слева единицу.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Существуют позиционные и непозиционные системы счисления. Мы пользуемся позиционными системами счисления, потому что они позволяют записать число в компактном виде и производить над числами все арифметические операции.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Получите числа, следующие за числами: а) 999_{16} , б) $AAFF_{16}$, в) $9AEF_{16}$.
2. Составьте и заполните следующую таблицу: впишите в каждый столбец по порядку числа системы счисления с основанием p .

$p = 10$	$p = 8$	$p = 5$	$p = 3$	$p = 2$	$p = 16$
0	0	0	0	0	0
1	1	1	1	1	1
...
...
17

3. Какое минимальное основание должна иметь система счисления, в которой могут быть записаны числа 28; 123; 1022; 189; AC12? Ответ запишите в виде цепочки чисел, разделённых символом «;».
4. Какие числа во всех системах счисления выглядят одинаково?
5. Запишите в развёрнутой форме число $N = 202, 33_4$.

§ 2. Перевод числа из произвольной системы счисления в десятичную. Перевод целого числа из десятичной системы счисления в произвольную

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

В задании к § 1 вы заполнили таблицу чисел в некоторых системах счисления. Она позволяет легко найти соответствие между числами в этих системах. Как вы думаете, может ли эта таблица оказаться полезной для вас?

- Есть ли здесь проблема? Сформулируйте основной вопрос урока. Сравните свою формулировку с авторской (с. 141 учебника).

РЕШЕНИЕ ПРОБЛЕМЫ

Теперь вы знаете, как представляются числа в разных системах счисления. Но как установить соответствие между этими числами? Посмотрите на составленную вами таблицу чисел в нескольких системах счисления. В ряде частных случаев она вам поможет. Но невозможно создать таблицу, в которой бы было любое интересующее вас число в любой системе счисления! Поэтому надо научиться переводить *любое* число из одной системы счисления в другую.

ПЕРЕВОД ЧИСЛА ИЗ ПРОИЗВОЛЬНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДЕСЯТИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

1. Число, записанное в системе счисления с основанием p , представить в развёрнутой форме.
2. Заменить цифры числа в данной системе на цифры в десятичной системе.
3. Вычислить значение полученного выражения в десятичной системе счисления. Результат — число, записанное в десятичной системе счисления.

Примеры

$$202_5 = 2 \cdot 5^2 + 0 \cdot 5^1 + 2 \cdot 5^0 = 25 + 0 + 2 = 52_{10};$$

$$11_{16} = 1 \cdot 16^1 + 1 \cdot 16^0 = 16 + 1 = 17_{10};$$

$$3A_{16} = 3 \cdot 16^1 + A \cdot 16^0 = 3 \cdot 16^1 + 10 \cdot 16^0 = 48 + 10 = 58_{10};$$

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13_{10};$$

$$22,6_7 = 2 \cdot 7^1 + 2 \cdot 7^0 + 6 \cdot 7^{-1} = 14 + 2 + 6/7 \approx 16,86_{10}.$$

ПЕРЕВОД ЦЕЛОГО ЧИСЛА ИЗ ДЕСЯТИЧНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ПРОИЗВОЛЬНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

1. Найти максимальную степень k основания p новой системы счисления, для которой p^k меньше данного десятичного числа.
2. Определить, сколько раз $[n]$ p^k «укладывается» в данном десятичном числе.
3. Записать $n \cdot p^k$ в качестве очередного слагаемого развёрнутой формы записи числа в новой системе счисления.
4. Получить разность данного десятичного числа и $n \cdot p^k$.
5. Для полученной разности повторять пункты № 1–4 до тех пор, пока разность не станет меньше основания p новой системы.
6. Записать последнее слагаемое как произведение полученной разности на p^0 .
7. Выписать коэффициенты при p^k в качестве цифр числа в новой системе так, чтобы их разряды соответствовали коэффициентам k . В разряды, соответствующие пропущенным k , записать нули.

Пример. Перевод в четверичную систему счисления числа 140_{10} .

$$\begin{array}{rcl}
 1 & 4 & 0_{10} = 2 \cdot 4^3 + 3 \cdot 4^1 + 0 \cdot 4^0 = 2030_4 \\
 - & 1 & 2 \quad 8 \\
 \hline
 & 1 & 2 \\
 - & 1 & 2 \\
 \hline
 & & 0
 \end{array}$$

Когда вы будете выполнять примеры, рассказывайте себе этот алгоритм. Рассуждайте (для приведённого примера) приблизительно так: «Основание новой системы счисления – 4. Мне надо найти максимальную степень k этого основания, для которой 4^k «умещается» в десятичном числе 140. $4^3 = 64$ меньше 140, а $4^4 = 256$ уже больше 140. Значит, это степень $k = 3$. Теперь посмотрю, сколько раз 4^3 «укладывается» в числе 140: 2 раза. *Запишу произведение $2 \cdot 4^3$ как первое слагаемое.* Найду произведение $2 \cdot 4^3 = 128$. Вычту 128 из данного числа 140. Разность равна 12. Она не меньше основания новой системы, значит, мне надо повторить предыдущие действия ещё раз – для числа 12. Теперь максимальная степень основания равна 1, 4^1 в числе 12 «укладывается» 3 раза. *Запишу в качестве второго слагаемого $3 \cdot 4^1$.* Далее из 12 я вычту $3 \cdot 4^1 = 12$. Разность 0 меньше 4, значит, далее поиск степени и вычисления производить не буду. *Запишу последнюю разность 0 на своё место: при 4^0 .* Теперь запишу коэффициенты на соответствующие места в числе. Замечу, что пропущен коэффициент при степени основания 2, в соответствующий разряд запишу 0».

А как следует поступить, если вы хотите перевести число из произвольной системы счисления в произвольную? Для этого можно использовать десятичную систему как «переводчика»: сначала перевести число из первой системы счисления в десятичную, а затем полученное десятичное число – во вторую систему.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Есть два алгоритма, которые позволяют перевести число из произвольной системы счисления в десятичную и целое число из десятичной системы в произвольную. Применяя их, можно переводить целые числа из любой системы счисления в любую, а «переводчиком» будет работать десятичная система счисления.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Укажите, какие целые числа следуют за числами:

- а) 101011_2 ; в) 567_8 ; д) $AABF_{16}$; ж) 133_4 ;
б) 28_9 ; г) 101111_2 ; е) 122212_3 ; з) 555_6 .

2. Какой цифрой заканчивается чётное двоичное число? Нечётное двоичное число? Какой цифрой может заканчиваться чётное троичное число?

3. Переведите в шестнадцатеричную систему счисления число 216_{10} . Сверьте своё решение с ответом:

$$216_{10} = 13 \cdot 16^1 + 8 \cdot 16^0 = D8_{16}.$$

4. Переведите числа:

- а) 10111_2 в шестеричную систему счисления;
б) $AO1_{16}$ в десятичную систему счисления;
в) 1333_4 в пятеричную систему счисления;
г) 512_8 в двоичную систему счисления;
д) 512_{10} в двоичную систему счисления.

§ 3. Переход между системами счисления, основания которых – степени двойки

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

На прошлом уроке вы научились переводить числа из одной системы счисления в другую и обнаружили, что посредником между системами счисления при переводе является десятичная система счисления. Но среди систем счисления есть такие, основания которых – степень двойки. Это объединяет их в особый класс. Они – «близкие родственники». Но в чём же проявляется «родство» этих систем и как оно нам может помочь? Если такие системы «ближе» друг к другу, чем к десятичной системе, то нужна ли она им как «посредник»?

- Как вы считаете, существуют ли особые приёмы перевода для данного случая? Сформулируйте проблему урока. Сравните свой вариант с авторским (с. 141 учебника).

РЕШЕНИЕ ПРОБЛЕМЫ

- Назовите системы счисления, основания которых – степень двойки.

Для чего мы прибегаем к помощи систем счисления с такими основаниями? Как вы знаете, для кодирования информации в компьютере применяется двоичная система счисления. Это обусловлено тем, что удобно использовать технические элементы, способные находиться в одном из двух состояний (есть ток – 1, нет тока – 0; низкое напряжение – 0, более высокое – 1 и т. д.). Но для человека громоздкие двоичные числа неудобны и непривычны в использовании, поэтому для записи адресов ячеек памяти используют восьмеричные и шестнадцатеричные числа.

Перевод чисел из одной системы счисления с основанием, равным степени числа 2, в другую такую систему, конечно, можно осуществлять через десятичную систему счисления. Но для таких систем счисления существует и другой, свой «посредник». Это двоичная система счисления. Давайте познакомимся со способом такого перевода. Он может показаться вам необычным, поскольку, в отличие от алгоритмов, приведённых в § 2, здесь ничего не придётся вычислять.

ПЕРЕВОД ВОСЬМЕРИЧНЫХ ЧИСЕЛ В ДВОИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

Для перевода восьмеричного числа в двоичную систему счисления надо каждую восьмеричную цифру представить отдельно в двоичной системе в виде трёх двоичных цифр (триады).

Примечание. Незначащие нули (перед первой единицей в целой части и после последней единицы в дробной части) можно опускать.

Примеры

$$\begin{array}{ccc} 011 & 111 & 101 \\ 3 & 7 & 5_8 \end{array} = \underbrace{11111101}_2$$

$$\begin{array}{cccccc} 010 & 101 & 001 & 001 & 000 & 110 \\ 2 & 5 & 1 & ,1 & 0 & 6_8 \end{array} = \underbrace{10101001,001000110}_2$$

Почему верен такой способ перевода? Минимальное *одинаковое* количество двоичных знаков, необходимое для представления любой восьмеричной цифры в двоичном коде, – три, поэтому для перевода мы записываем каждую восьмеричную цифру тремя двоичными. Пусть надо перевести в двоичную систему счисления число 63_8 . По предложенному правилу у нас должно получиться число 110 011.

Представим наше восьмеричное число в развёрнутой форме и воспользуемся равенством $2^3 = 8$:

$$63_8 = 6 \cdot 8^1 + 3 \cdot 8^0 = 6 \cdot (2^3)^1 + 3 \cdot (2^3)^0 = 6 \cdot 2^3 + 3 \cdot 2^0.$$

Переведём, как нам указывает наше правило, отдельно каждую восьмеричную цифру в двоичную систему, представим каждое получившееся двоичное число в развёрнутой форме и раскроем скобки:

$$\begin{aligned} 63_8 &= 6 \cdot 8^1 + 3 \cdot 8^0 = 6 \cdot (2^3)^1 + 3 \cdot (2^3)^0 = 6 \cdot 2^3 + 3 \cdot 2^0 = (110) \cdot 2^3 + (011) \cdot 2^0 = \\ &= [1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0] \cdot 2^3 + [0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0] \cdot 2^0 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + \\ &+ 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0. \end{aligned}$$

Но что мы получили? Это же развёрнутая форма двоичного числа! Значение этого двоичного числа равно значению числа 63_8 . Выпишем двоичное число по его развёрнутой форме: 110011. Оно совпадает с полученным по правилу перевода.

Приведём таблицу соответствия восьмеричных цифр и двоичных чисел.

Восьмеричная цифра	0	1	2	3	4	5	6	7
Двоичное число	000	001	010	011	100	101	110	111

ПЕРЕВОД ДВОИЧНЫХ ЧИСЕЛ В ВОСЬМЕРИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

Для перевода двоичного числа в восьмеричную систему счисления надо двоичное число разделить на триады и каждую триаду заменить на соответствующую ей восьмеричную цифру.

Примечание. Целая часть делится на триады справа налево, а дробная – слева направо. Если при делении целой части на триады окажется, что в последней (левой) выделяемой группе меньше трёх цифр, никаких дополнительных действий не требуется. Если не хватает цифр при выделении триад в дробной части, необходимо дописать в конец дроби столько нулей, сколько цифр не хватает.

Пример

Направления деления на триады
целой части дробной части

$$10101,01001_2 = 10101,01001\mathbf{0}_2 = 25,22_8$$

Выделен O , который мы добавили до триады в конце дроби.

ПЕРЕВОД ШЕСТНАДЦАТЕРИЧНЫХ ЧИСЕЛ В ДВОИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

Для перевода шестнадцатеричного числа в двоичную систему счисления надо каждую шестнадцатеричную цифру представить соответствующей ей четвёркой двоичных цифр – *тетрадой*.

Примечание. Незначащие нули (перед первой единицей в целой части и после последней единицы в дробной части) можно опускать.

Это правило основано на том, что минимальное *одинаковое* количество двоичных знаков, необходимое для представления любой шестнадцатеричной цифры в двоичном коде, – четыре, поэтому для перевода мы записываем каждую шестнадцатеричную цифру четырьмя двоичными.

Примеры

$$\begin{array}{ccc} 0011 & 0010 & 0101 \\ 3 & 2 & 5_{16} \end{array} = 1100100101_2$$

$$\begin{array}{ccccccc} 0010 & 0001 & & 0001 & 0010 & & \\ 2 & 1 & , & 1 & 2_{16} & = & 100001,0001001_2 \end{array}$$

- Покажите на примере, аналогично тому, как это сделано выше для восьмеричной системы счисления, почему это правило верное.

Приведём таблицу соответствия шестнадцатеричных цифр и двоичных чисел.

Шестнадцатеричная цифра	0	1	2	3	4	5	6	7
Двоичное число	0000	0001	0010	0011	0100	0101	0110	0111
Шестнадцатеричная цифра	8	9	A	B	C	D	E	F
Двоичное число	1000	1001	1010	1011	1100	1101	1110	1111

ПЕРЕВОД ДВОИЧНЫХ ЧИСЕЛ В ШЕСТИНАДЦАТЕРИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ

Правило перевода:

Для перевода двоичного числа в шестнадцатеричную систему счисления надо разделить двоичное число на тетрады и каждую тетраду заменить на соответствующую ей шестнадцатеричную цифру.

Примечание. Целая часть делится на тетрады справа налево, а дробная – слева направо. Если при делении целой части на тетрады окажется, что в последней (левой) выделяемой группе меньше четырёх цифр, никаких дополнительных действий не требуется. Если не хватает цифр при выделении тетрад в дробной части, необходимо дописать в конец дроби столько нулей, сколько цифр не хватает.

Пример

$$\begin{array}{ccccccc} 1 & & & & & & \\ 1 & 1011 & 1011 & , & 1011 & 1000_2 & = 1BB,BB_{16} \end{array}$$

Заметим, что приписывание 0 справа к двоичному числу увеличивает его в 2 раза, так же как приписывание к десятичному – в 10 раз, к шестнадцатеричному – в 16 раз и т. п. Это можно использовать для определения шестнадцатеричного выражения комбинаций двоичных цифр.

Двоичное число	Действия	Десятичное число	Шестнадцатеричная цифра
1 0 0 : 0	$4 \cdot 2 + 0$	8	8
1 0 0 : 1	$4 \cdot 2 + 1$	9	9
1 0 1 : 0	$5 \cdot 2 + 0$	10	A
1 0 1 : 1	$5 \cdot 2 + 1$	11	B
1 1 0 : 0	$6 \cdot 2 + 0$	12	C
1 1 0 : 1	$6 \cdot 2 + 1$	13	D
1 1 1 : 0	$7 \cdot 2 + 0$	14	E
1 1 1 : 1	$7 \cdot 2 + 1$	15	F

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Вы узнали, что переводить числа между системами счисления, основания которых – степени двойки, можно, используя в качестве «посредника» двоичную систему счисления.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Переведите числа в восьмеричную систему счисления: а) $123,13_{16}$; б) $1111,111_4$; в) $EDA.DA_{16}$.
2. Переведите числа в шестнадцатеричную систему счисления: а) $123,123_4$; б) $1515,21_8$; в) $100,0E_{16}$.
3. Переведите числа в четверичную систему счисления: а) $ABC,01_{16}$; б) $111,001_8$; в) $111,001_{16}$.
4. Предложите правило перевода чисел из четверичной в восьмеричную систему счисления.

§ 4. Сложение и вычитание чисел в произвольных системах счисления

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Итак, мы с вами «путешествуем» по миру чисел. Вы узнали много нового о числах и умеете переводить целые числа из одной позиционной системы счисления в другую. Мир чисел велик, числа окружают нас повсюду, без них очень сложно обходиться. Но зачем нам нужны числа? Правильно, чтобы производить расчёты. В десятичной системе вы давно научились этому. А что происходит в других системах? Существуют ли там свои правила? Чем арифметические действия в произвольной системе отличаются от действий в десятичной системе?

- Как вы считаете, какая проблема в этой ситуации? Сформулируйте её. Сравните свой вариант с авторским (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните из курса математики, как производится сложение и вычитание десятичных чисел.

РЕШЕНИЕ ПРОБЛЕМЫ

Давайте расширим умения обращения с числами в разных системах счисления и добавим к ним опыт сложения и вычитания целых чисел в произвольной системе счисления. Арифметические действия во всех позиционных системах счисления выполняются по одним и тем же правилам. С этими правилами вы давно знакомы по десятичной системе счисления. Это хорошая новость: никаких новых правил! Но при этом следует помнить о том, что в других позиционных системах другие основания, к которым вы ещё не привыкли. Если в десятичной системе число 10 следует за числом 9, то, например, в пятеричной системе 10 идёт после 4, а в двоичной – после 1. От этого зависит, например, сколько единиц мы переносим в следующий разряд при сложении чисел. Следует быть внимательными при выполнении операций над числами, помнить, какое основание у данной системы.

Отметим важный момент: перед тем, как произвести любую арифметическую операцию, числа надо записать в одной и той же системе счисления.

СЛОЖЕНИЕ

Правила сложения чисел в произвольной системе счисления аналогичны правилам сложения чисел в десятичной системе счисления.

Это означает, что, во-первых, сложение чисел в некотором разряде производится согласно таблице сложения в данной системе счисления (с основа-

нием p). Во-вторых, если при сложении цифр в некотором разряде получается значение, большее старшей цифры в данной системе счисления, в соответствующий разряд результата надо записать разность между этим значением и значением 10_p в данной системе. Образовавшуюся цифру переноса нужно учитывать при сложении в следующем разряде.

Таблицу сложения в десятичной системе вы знаете. Приведём таблицы сложения в двоичной (рис. 2.1) и восьмеричной (рис. 2.2) системах. Цифры-слагаемые находятся в строках и столбцах, результаты сложения – на их пересечении: в ячейках таблицы.

Таблица сложения в двоичной системе счисления:

+	0	1
0	0	1
1	1	10

Рис. 2.1

Таблица сложения в восьмеричной системе счисления:

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Рис. 2.2

- Какую закономерность вы видите в таблицах сложения?

Аналогично строятся таблицы сложения для других систем счисления.

Рассмотрим правило сложения на примерах.

Пример 1

Сложение целых двоичных чисел:

$$\begin{array}{r}
 \begin{array}{cccccc}
 & * & & * & & \\
 + & 1 & 1 & 0 & 0 & 1_2 \\
 & & 1 & 1 & 0 & 1_2 \\
 \hline
 1 & 0 & 0 & 1 & 1 & 0_2
 \end{array}
 \end{array}$$

Пример 2

Сложение дробных двоичных чисел:

$$\begin{array}{r}
 \begin{array}{cccccc}
 & * & * & * & * & \\
 + & 1 & 1 & 1 & 1, & 1 & 1_2 \\
 & & 1 & 1 & 1, & 1_2 & \\
 \hline
 1 & 0 & 1 & 1 & 1, & 0 & 1_2
 \end{array}
 \end{array}$$

Пример 3
Сложение целых
восьмеричных чисел:

$$\begin{array}{r}
 \cdot \\
 \begin{array}{r}
 1 \quad 7_8 \\
 + 7 \quad 3_8 \\
 \hline
 1 \quad 1 \quad 2_8
 \end{array}
 \end{array}$$

Точки обозначают перенос единицы в следующий разряд.

Обратите внимание на пример сложения восьмеричных чисел. Сложение в младшем разряде в восьмеричной системе цифр 3 и 7 здесь даёт не 10, как в десятичной системе, а 12_8 . Из этого числа вычитается 10_8 . Поэтому цифрой результата будет 2. При этом переносится единица в старший разряд.

Для двоичной системы (см. примеры 1 и 2) сложение 1 и 1 даёт 10_2 , поэтому цифрой результата будет 0, при этом в старший разряд переносится единица.

Можно пользоваться следующим приёмом сложения. Цифры в разряде складываются в десятичной системе, а результат сложения затем переводится в нужную систему счисления. Вычислять в десятичной системе нам привычнее. В рассмотренном примере 3 сложим числа 7 и 3 в десятичной системе: получим 10 в десятичной системе. Переведём 10 в восьмеричную систему: получим 12_8 .

Теперь, зная, как суммируются числа в двоичной системе счисления, можно воспользоваться одним полезным алгоритмом для перевода в десятичную систему счисления больших двоичных чисел, в записи которых много единиц и мало нулей, например числа 111110101_2 . Если делать так, как вы научились, то придётся произвести следующие действия: $111110101_2 = 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^0$. Шансов ошибиться, производя эти действия, много. И времени вы потратите много. Сделаем лучше так. Представьте себе, что перед вами лист тетради в клетку (таблица, рис. 2.3).

Запишем данное число, поставив над каждым разрядом степень двойки.

Получим из данного числа новое заменой 0 на 1, а 1 на 0. Такую операцию называют *инвертированием*.

Сложим данное и инвертированное числа. В результате получим число, состоящее из одних единиц.

Прибавим к сумме 1. В результате все единицы заменятся на нули, при этом появится ещё один старший разряд с единицей. Для нашего примера 9-разрядного числа получим $2^9 = 512$.

Теперь будем считать, поднимаясь вверх по таблице (см. рис. 2.3).

В верхней строчке получим ответ: 501.

9	8	7	6	5	4	3	2	1	0	
	1	1	1	1	1	0	1	0	1	$511 - 10 = 501$
+	0	0	0	0	0	1	0	1	0	$2^9 + 2^1 = 10$
	1	1	1	1	1	1	1	1	1	$512 - 1 = 511$
+									1	
1	0	0	0	0	0	0	0	0	0	$2^9 = 512$

Рис. 2.3

Замечание. Отметим, что рассмотренные на рис. 2.3 операции инвертирования и прибавления единицы используются для представления отрицательных чисел в компьютере. С ними вы сможете подробно познакомиться в § 8.

Обратим особое внимание на сложение шестнадцатеричных чисел. Оно не сложнее других, просто складывать буквы как-то непривычно. Складывают, конечно, не буквы, а их численные значения: А – 10; В – 11; С – 12; D – 13; Е – 14; F – 15. Складывать будем в десятичной системе, а результат записывать в шестнадцатеричной.

Пример 4

$$\begin{array}{r}
 \begin{array}{r}
 2 \quad E \quad 2 \\
 + \quad A \quad 5 \quad 4 \\
 \hline
 D \quad 3 \quad 6
 \end{array}
 \end{array}$$

1-й разряд (16^0): $(2 + 4)_{16}$; $(2 + 4)_{10} = 6_{10} = 6_{16}$;
 2-й разряд (16^1): $(E + 5)_{16}$; $(14 + 5)_{10} = 19_{10} = 13_{16}$;
 пишем 3, а 1 переносим в 3-й разряд;
 3-й разряд (16^2): $(2 + A + 1)_{16}$; $(2 + 10 + 1)_{10} = 13_{10} = D_{16}$.

- Выполните примеры 5, 6, 7 самостоятельно и сверьте полученный результат с ответом.

Пример 5

Сложение

пятиричных чисел:

$$\begin{array}{r}
 \begin{array}{r}
 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
 + \quad 4 \quad 4 \quad 2 \quad 3 \quad 4 \\
 \quad \quad 3 \quad 3 \quad 3 \quad 3 \\
 \hline
 1 \quad 0 \quad 3 \quad 1 \quad 2 \quad 2
 \end{array}
 \end{array}$$

Пример 6

Сложение

шестнадцатеричных чисел:

$$\begin{array}{r}
 \begin{array}{r}
 1 \quad 1 \quad \quad 1 \quad 1 \\
 + \quad E \quad D \quad A, \quad 9 \quad 8 \\
 \quad 1 \quad F \quad 1, \quad A \quad B \\
 \hline
 1 \quad 0 \quad C \quad C, \quad 4 \quad 3
 \end{array}
 \end{array}$$

Пример 7

Сложение
восьмеричных чисел:

$$\begin{array}{r}
 1 1 1 \\
 + 5 7, 5 \\
 \hline
 1 5 3, 3
 \end{array}$$

ВЫЧИТАНИЕ

Вычитание чисел в любых позиционных системах счисления выполняется по тем же правилам, что и в десятичной системе счисления.

Рассмотрим это на примере вычитания 16-разрядных чисел.

Пример 8

$$\begin{array}{r}
 - 6 9 \\
 2 A \\
 \hline
 3 F
 \end{array}$$

Из 2-го разряда (его вес — 16^1) занимаем 1 (то есть $16^1 = 10_{16}$); в 1-м разряде получается:
 $(9 + 10 - A)_{16} = (9 + 16 - 10)_{10} = 15_{10} = F_{16}$.
 Производим действия во 2-м разряде: $(5 - 2)_{16} = 3_{16}$.

Рассмотрим теперь подробно схему вычитания чисел для общего случая, когда требуется занимать из старших разрядов.

- Вспомните, как вы производите заёмы при вычитании в десятичной системе счисления.

Схема основана на том, что любую степень основания p^n в любой системе счисления можно представить как сумму младших степеней основания, где коэффициент при любой степени, кроме самой младшей из используемых, всегда на единицу меньше основания, а коэффициент при самой младшей степени равен основанию.

Пример 9

В двоичной системе:

$$\begin{aligned}
 2^4 &= 2 \cdot 2^3 \\
 2^4 &= 1 \cdot 2^3 + 2 \cdot 2^2 \\
 2^4 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 2 \cdot 2^1 \\
 2^4 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 2 \cdot 2^0
 \end{aligned}$$

В семеричной системе:

$$7^5 = 6 \cdot 7^4 + 6 \cdot 7^3 + 7 \cdot 7^2$$

Пример 10

$$\begin{array}{cccc} & 3 & 2 & 1 & 0 \\ & 1 & 0 & 0 & 0_2 \\ - & & & 1 & 1_2 \\ \hline & & 1 & 0 & 1_2 \end{array}$$

$$1000_2 - 11_2 = 101_2$$

Мы видим, что в разрядах с 1-го по 3-й невозможно произвести вычитание – надо занять единицу в 4-м разряде (то есть 2^3). Поэтому разложим 2^3 до 2^0 : $2^3 = 1 \cdot 2^2 + 1 \cdot 2^1 + 2 \cdot 2^0$ (распределим единицу в 4-м разряде, то есть 2^3 , по младшим разрядам). Коэффициенты в этом разложении будем учитывать при вычитании в соответствующих разрядах. В 1-м разряде произведём действия: $(2 + 0) - 1 = 1$ – пишем 1 в числе-результате. Во 2-м разряде: $(1 + 0) - 1 = 0$ – пишем 0. В 3-м разряде: $1 + 0 = 1$ – пишем 1. В 4-м разряде была одна единица, но мы её заняли, и там ничего не осталось.

Пример 11

$$\begin{array}{cccccc}
 2 & 1 & 0 & -1 & -2 & -3 \\
 1 & 0 & 0 & 0 & 0 & 1_2 \\
 & & 1 & 1 & 1 & 1_2 \\
 \hline
 & 1 & 0 & 0 & 1 & 0_2
 \end{array}$$

$$100,001_2 - 1,111_2 = 10,01_2.$$

Занимаем двойку во второй степени;
раскладываем до 2^{-2} : $2^2 = 1 \cdot 2^1 + 1 \cdot 2^0 +$
 $+ 1 \cdot 2^{-1} + 2 \cdot 2^{-2}$; учитываем коэффициенты
при вычитании.

Пример 12

$$\begin{array}{r} 210 \\ - 706_8 \\ \hline 627_8 \end{array}$$

Занимаем 8^2 , раскладываем до 8^0 : $8^2 = 7 \cdot 8^1 + 8 \cdot 8^0$, в 1-м разряде (8^0) учитываем коэффициент 8: $8 + 6 = 14$; после вычитания остаётся $14 - 7 = 7$. Во 2-м разряде: $(7 + 0) - 5 = 2$. В 3-м разряде после заёма остаётся 6.

$$706_8 - 57_8 = 627_8$$

Вспомните, как вы занимаете при вычитании в десятичной системе счисления: вы, на самом деле, применяете рассмотренный приём, но делаете это уже автоматически. Таким же автоматическим станет для вас процесс вычитания в произвольной системе счисления после приобретения некоторого опыта. Вы уже не будете задумываться о каждом конкретном шаге схемы, рассмотренной здесь так подробно.

- Подумайте, для решения каких ещё задач вам может пригодиться возможность рассмотренного представления степени числа.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Арифметические действия производятся по одинаковым правилам во всех позиционных системах счисления. При выполнении арифметических действий необходимо помнить, каково основание системы, в которой они производятся, и учитывать это. Сложение и вычитание нужно выполнять над числами, представленными в одной и той же системе счисления.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Выполните сложение.

а) $754_8 + 341_8$;

б) $12A,8B_{16} + BBA,45_{16}$;

в) $467_8 + 123_{16}$. Результат представьте в шестнадцатеричной системе счисления.

2. Выполните вычитание.

а) $10100101_2 - 111111_2$;

б) $1011101,011_4 - 222,23_4$;

в) $ABC,DE_{16} - 1534,5_8$. Результат представьте в восьмеричной системе счисления.

§ 5. Перевод правильной десятичной дроби в произвольную систему счисления

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Как перевести из десятичной системы счисления в произвольную целое число, вы теперь знаете. Но мир чисел не ограничивается только целыми числами.

• Как вы думаете, «секреты» работы с какими числами вы сейчас узнаете? Сформулируйте основной вопрос урока. Сравните свою формулировку с авторской (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните из курса математики, какая десятичная дробь называется правильной.

РЕШЕНИЕ ПРОБЛЕМЫ

Правило перевода дроби:

1. Нарисовать вертикальную черту.
2. Расположить переводимое число так, чтобы O с запятой были слева от вертикальной черты, а дробная часть — справа.
3. Умножить только дробную часть на основание p системы счисления. Полученную целую часть записать слева от черты, дробную — справа.
4. Выполнять пункт № 3 до тех пор, пока часть, записываемая справа от черты, не станет равной O . Справа от черты при этом будут появляться цифры от O до $p - 1$. Может так случиться, что в правой части O никогда не получится. В этом случае умножение продолжать до необходимой степени точности или до появления периодической дроби.
5. Выписать сверху вниз цифры, расположенные справа от черты (в примере ниже выделены красным), первое из них — ноль отделить запятой. Эти цифры представляют собой искомую дробь, записанную в новой системе счисления.

Помните: умножайте только числа, записанные справа от черты, а в результате записывайте цифры, получающиеся слева от черты.

Примеры

Перевод в двоичную систему счисления дроби $0,125_{10}$:

$$\begin{array}{r|l}
 0, & \times 125 \\
 \hline
 0 & \times 250 \\
 \hline
 0 & \times 500 \\
 \hline
 1 & 000
 \end{array}$$

$$0,125_{10} = 0,001_2$$

Перевод в троичную систему счисления дроби $0,125_{10}$:

$$\begin{array}{r|l}
 0, & \times 125 \\
 \hline
 0 & \times 375 \\
 \hline
 1 & \times 125 \\
 \hline
 0 & \times 375 \\
 \hline
 1 & \times 125 \\
 \hline
 & \times 3
 \end{array}$$

$$0,125_{10} = 0,01(01)_3 \text{ — получили периодическую дробь.}$$

Отметим, что в случае перевода правильной десятичной дроби в шестнадцатеричную систему счисления дробная часть каждый раз умножается на двузначное число (16), по правилу умножения на двузначное число; то есть очередная дробная часть умножается на 6 и на 1, результаты умножения записываются со сдвигом и складываются. Целая часть результата сложения представляет собой очередную цифру искомой шестнадцатеричной дроби. Не забывайте цифры целой части, большие 9, записывать в виде букв А–F шестнадцатеричной системы счисления. *Напоминание:* умножайте только числа, записанные справа от черты (в примере ниже выделены красным), а в результат записывайте цифры, получающиеся слева от черты.

Пример

Перевод десятичной дроби $0,124$ в шестнадцатеричную систему счисления:

	0,	1	2	4
			1	6
+		7	4	4
	1	2	4	
	1	9	8	4
			1	6
+	5	9	0	4
	9	8	4	
	F	7	4	4
			1	6
+	4	4	6	4
	7	4	4	
	B	9	0	4

$$0,124_{10} \approx 0,1FB_{16}$$

Правило перевода смешанного числа:

Чтобы перевести из десятичной системы счисления в произвольную смешанное десятичное число, необходимо отдельно перевести в новую систему счисления целую и дробную части и затем сложить их.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Алгоритм перевода правильной десятичной дроби в произвольную систему счисления отличается от алгоритма перевода целого числа.

При переводе в шестнадцатеричную систему счисления необходимо следовать правилу умножения на двузначное число и помнить, что, если целая часть результата сложения превышает 9, она записывается в виде соответствующей буквы (A–F).

В смешанном числе отдельно переводятся целая и дробная части. Для получения окончательного ответа результаты перевода целой части и правильной десятичной дроби суммируются.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Переведите правильную десятичную дробь 0,125 в следующие системы счисления:

- | | |
|-----------------|-----------------------|
| а) пятеричную; | в) восьмеричную; |
| б) четверичную; | г) шестнадцатеричную. |

2. Переведите число 122,55 из десятичной системы счисления в следующие системы счисления:

- | | |
|-----------------|-----------------------|
| а) пятеричную; | в) восьмеричную; |
| б) четверичную; | г) шестнадцатеричную. |

Проверь себя**Задание 1**

Определите, какое число следует за числом 377_8 .

Задание 2

Переведите число 12334 в шестеричную систему счисления.

Задание 3

Переведите из восьмеричной системы счисления в шестнадцатеричную систему счисления число $735,12_8$.

Задание 4

Выполните сложение двоичных чисел:

$$100111,1_2 + 111011,011_2.$$

Задание 5

Выполните вычитание двоичных чисел:

$$10010101101_2 - 1110011111_2.$$

Задание 6

Выполните вычитание восьмеричных чисел:

$$213 - 67_8.$$

Задание 7

Переведите из десятичной системы счисления правильную десятичную дробь 0,75 в шестеричную систему счисления.

ПРИМЕНЯЕМ ЗНАНИЯ (необходимый уровень)

Задание 1

Запишите целое число, следующее за данным:

- а) ABF_{16} ;
- б) 1233_4 ;
- в) 100111_2 .

Задание 2

Переведите:

- а) в десятичную систему счисления числа $255,5_6$; 10111_2 ;
- б) из десятичной системы счисления в шестеричную и в шестнадцатеричную системы счисления число 117;
- в) из восьмеричной системы счисления в шестнадцатеричную систему счисления число 456_8 .

Задание 3

Найдите сумму двоичных чисел $110110,11_2$ и $11101,111_2$.

Задание 4

Найдите разность двоичных чисел 1110011_2 и 111101_2 .

Задание 5

Переведите число $111,2233_4$ из четверичной системы счисления в шестнадцатеричную систему счисления.

Задание 6

Выпишите целые числа от 212_3 до 1000_3 в троичной системе счисления.

Задание 7

Какие целые числа предшествуют числам: 200_6 ; 1000_2 ; 770_8 ; 400_5 ?

Задание 8

Переведите в шестеричную систему счисления десятичную дробь 0,25.

ПРИМЕНЯЕМ ЗНАНИЯ (повышенный уровень)

Задание 1

Переведите число 325_6 в пятеричную систему счисления.

Задание 2

Найдите сумму чисел 213_4 и 111_8 . Результат запишите в четверичной системе счисления.

Задание 3

Найдите разность чисел 111001101_2 и 500_9 . Результат запишите в восьмеричной системе счисления.

Задание 4

Найдите сумму чисел 175_{16} и 116_{16} . Ответ запишите в восьмеричной системе счисления.

Задание 5

Переведите число 665_9 из восьмеричной системы счисления в троичную систему счисления.

Задание 6

Какое наибольшее десятичное число можно записать двумя цифрами в: а) двоичной; б) восьмеричной; в) шестнадцатеричной системах счисления?

Задание 7

Составьте таблицы сложения однозначных чисел в троичной и четверичной системах счисления.

Задание 8

Дед, Бабка, Внучка и Жучка собирали в саду яблоки. Когда Мышка спросила, кто сколько собрал, каждый дал ответ в своей системе счисления. Дед назвал число 55, Бабка – 44, Внучка – 22, а Жучка – 11. Когда Дед добавил, что если бы каждый положил в свою корзину ещё только одно яблоко, то число яблок в каждой корзине можно было бы записать как 100, для Мышки уже не было секретом, сколько всего яблок собрала дружная семья. Какое количество яблок насчитала Мышка?

Задание 9

Переведите в восьмеричную систему счисления числа $34,125_{10}$ и $118,25_{10}$ и затем сложите их. Ответ запишите в шестнадцатеричной системе счисления.

ПРИМЕНЯЕМ ЗНАНИЯ (повышенный уровень)**Задание 1**

В некоторой системе счисления десятичное число 74 записывается как 202. Чему равно основание этой системы счисления?

Задание 2

В спортивной секции плаванием занимаются 210 детей. Среди них 120 мальчиков и 50 девочек. В какой системе счисления записаны данные?

Задание 3

Даны три числа в шестнадцатеричной системе счисления $A1_{16}$, 10_{16} и $B1_{16}$. Найдите их сумму. Ответ запишите в восьмеричной системе счисления.

Задание 4

Даны три числа в восьмеричной системе счисления: 175_8 , 124_8 , 116_8 . Найдите их сумму. Ответ запишите в шестнадцатеричной системе счисления.

Задание 5

В какой системе счисления справедливо равенство: $98 + 89 = 121$?

Задание 6

В какой системе счисления справедливо равенство: $35 + 21 = 100$?

Задание 7

Найдите сумму шестнадцатеричных чисел $A1$, $A3$, $A5$, $A7$. Ответ запишите в восьмеричной системе счисления.

Задание 8

Запишите число 255 в системе счисления с основанием 254.

Задание 9

Школьный калькулятор работает в четверичной системе счисления и может выводить на экран максимум четырёхзначное число. Каким будет максимальное десятичное число, которое можно отобразить на экране этого калькулятора?

Задание 10

Переведите в шестнадцатеричную систему счисления числа $34,125_{10}$ и $118,25_{10}$ и затем сложите их. Ответ запишите в восьмеричной системе счисления.

Задание 11

В классе 111100_2 процентов девочек и 1100_2 мальчиков. Сколько учеников в классе?

Итоговая проверочная работа

Задание 1

- а) Расположите числа в порядке убывания: 1000110_2 ; 456_8 ; 251_{10} .
- б) Сложите числа 10001010_2 и 1233_4 . Результат запишите в восьмеричной системе счисления.
- в) Найдите разность чисел 100001001_2 и 321_8 . Результат запишите в двоичной системе счисления.
- г) Переведите число $234,08_{10}$ в пятеричную систему счисления.

Задание 2

- а) Запишите в десятичной системе счисления:
- наибольшее трёхзначное число в восьмеричной системе счисления;
 - наибольшее трёхзначное число в шестнадцатеричной системе счисления.
- б) Найдите сумму и разность чисел 567_8 и 1021_6 . Ответ запишите в восьмеричной системе счисления.
- в) Переведите числа $234,55_{10}$ и $123,75_{10}$ в шестеричную систему счисления с точностью до второго знака после запятой и найдите их разность.

Задание 3

- а) Вычислите значение выражения: $123_8 + AB_{16} - 81_{10}$. Ответ запишите в восьмеричной системе счисления.
- б) Переведите числа $234,55_{10}$ и $123,75_{10}$ в шестнадцатеричную систему счисления с точностью до второго знака после запятой и найдите их сумму и разность.
- в) Расставьте вместо знаков «?» знаки арифметических операций так, чтобы были верны следующие равенства в двоичной системе счисления:
- $1100_2 ? 11_2 ? 100_2 = 100000_2$;
 - $1100_2 ? 10_2 ? 10_2 = 100_2$;
 - $1100_2 ? 10_2 ? 10_2 = 110000_2$.

§ 6. Деление и умножение в позиционных системах счисления

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Из четырёх арифметических действий над числами в системах счисления вы освоили два: сложение и вычитание. Конечно, при желании можно было бы обходиться только ими: умножение заменять сложением, а деление – вычитанием. Но на практике умение умножать и делить сильно сокращает время вычислений.

- Как вы думаете, какие полезные навыки вы будете сейчас приобретать? Сформулируйте тему урока. Сравните свой вариант с авторским (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Назовите характеристики позиционной системы счисления. (§ 1)

Как производится операция сложения в произвольной системе счисления? (§ 4)

Как производится операция вычитания в произвольной системе счисления? (§ 4)

РЕШЕНИЕ ПРОБЛЕМЫ

УМНОЖЕНИЕ

Умножение чисел в любой системе счисления (двоичной, троичной, восьмеричной и т. д.) выполняется по тем же правилам, что и в десятичной системе. Если при умножении цифр в некотором разряде получается значение, большее старшей цифры в данной системе счисления, в соответствующий разряд результата надо записать остаток от деления этого значения на 10_p в данной системе. Образовавшуюся цифру переноса (частное от деления на 10_p) нужно учитывать при вычислении в следующем разряде.

Можно пользоваться следующим приёмом умножения. Цифры в разряде умножаются в десятичной системе, а результат умножения затем переводится в нужную систему счисления.

В двоичной системе счисления операция умножения значительно упрощается, так как умножаются только цифры 0 и 1. Причём умножение на 1 первого сомножителя означает просто его копирование с соответствующим сдвигом по разрядам, без вычислений, а умножение на 0 вообще пропускается.

Пример 2

$$\begin{array}{r} 1 1 0 1 \\ \times 1 1 0 0 \\ \hline 1 1 0 1 \\ 1 1 0 1 \\ \hline 1 0 0 1 1 0 0 \end{array}$$

$$1101_2 \cdot 1100_2 = 10011100_2$$

$$\begin{array}{r} \times 101,01 \\ \hline 10101 \\ 1010100 \\ \hline 101011101 \end{array}$$

Нахождение произведения восьмеричных чисел:

$$\begin{array}{r} 13 \\ 725_8 \\ \times 206_8 \\ \hline 5376_8 \\ + 1652 \\ \hline 172576_8 \end{array}$$

При умножении в 1-м разряде в десятичной системе счисления получается $30_{10} = 36_6$, поэтому в 1-м разряде пишем 6, а 3 переносим во 2-й разряд. При следующем умножении получаем $15(6 \cdot 2 + 3 = 15)$ в десятичной системе счисления, $15_{10} = 17_6$, поэтому во 2-м разряде пишем 7, а 1 переносим в следующий разряд и т. д.

$$725_8 \cdot 206_8 = 17\,2576_8$$

ДЕЛЕНИЕ

Деление чисел в любой системе счисления производится так же, как и в десятичной системе счисления. Можно пользоваться при выполнении действий десятичной системой, результаты переводить в нужную систему.

Пример 4

Деление чисел 11001_2 и 101_2 :

$$\begin{array}{cccc|ccc}
 - & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
 & 1 & 0 & 1 & & & 1 & 0 & 1 \\
 \hline
 & & & 1 & 0 & 1 & & & \\
 & - & & 1 & 0 & 1 & & & \\
 & & & \hline
 & & & & 0 & & & &
 \end{array}$$

Пример 5

Деление чисел 111111_2 и 10101_2 :

$$\begin{array}{r} \begin{array}{cccccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & & 1 & 1 & & & \end{array} \\ \hline \begin{array}{cccccc|cccc} & 1 & 0 & 1 & 0 & 1 & & & & & \end{array} \\ - \begin{array}{cccccc|cccc} & 1 & 0 & 1 & 0 & 1 & & & & & \end{array} \\ \hline \begin{array}{cccccc|cccc} & & & & & 0 & & & & & \end{array} \end{array}$$

Пример 6Деление чисел 101101_2 и 10010_2 :

$$\begin{array}{r}
 101101_2 : 10010_2 = 1001_2 \\
 \underline{10010} \\
 10001 \\
 \underline{10000} \\
 10010 \\
 \underline{10010} \\
 0
 \end{array}$$

Пример 7Деление чисел 23363_8 и 73_8 :

$$\begin{array}{r}
 23363_8 : 73_8 = 321_8 \\
 \underline{166} \\
 456 \\
 \underline{447} \\
 73 \\
 \underline{73} \\
 0
 \end{array}$$

В числе 233_8 поместится

$2 \cdot 73_8 = 166_8.$

В числе 456_8 поместится

$5 \cdot 73_8 = 447_8.$

$456_8 - 447_8 = 7_8$

В числе 73_8 поместится

$1 \cdot 73_8 = 73_8.$

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ НАД ШЕСТНАДЦАТЕРИЧНЫМИ ЧИСЛАМИ**Примеры**

Сложение:

$$\begin{array}{r}
 2 \quad E \quad 2 \\
 A \quad 5 \quad 4 \\
 \hline
 D \quad 3 \quad 6
 \end{array}$$

1-й разряд: $(2 + 4)_{16} = (2 + 4)_{10} = 6_{10} = 6_{16};$ 2-й разряд: $(E + 5)_{16} = (14 + 5)_{10} = 19_{10} = 13_{16};$

пишем 3, а 1 переносим в 3-й разряд:

$(2 + A + 1)_{16} = (2 + 10 + 1)_{10} = 13_{10} = D_{16}.$

Вычитание:

$$\begin{array}{r}
 6 \quad 9 \\
 2 \quad A \\
 \hline
 3 \quad F
 \end{array}$$

1-й разряд: занимаем $1 \cdot 16^1 (16_{10})$ из 2-го разряда;

$(9 + 16 - A)_{16} = (9 + 16 - 10)_{10} = 15_{10} = F_{16};$

$2 - 2_{16} = 0_{16}.$

Умножение:

$$\begin{array}{r}
 2 \\
 2 \text{ E} \\
 \times \quad 3 \\
 \hline
 8 \text{ A}
 \end{array}$$

1-й разряд: $(E \cdot 3)_{16} = (14 \cdot 3)_{10} = 42_{10} = (2 \cdot 16^1 + 10 \cdot 16^0)_{10} = 2A_{16}$;
 пишем A, 2 переносим в следующий разряд;
 2-й разряд: $(2 \cdot 3 + 2) = 8_{16}$.

Деление:

$$\begin{array}{r}
 \text{B} \ 4 \mid 4 \\
 - \ 8 \quad \quad \mid 2 \ D \\
 \hline
 3 \ 4 \\
 - \ 3 \ 4 \\
 \hline
 0
 \end{array}$$

$B_{16} = 11_{10}$; $(11:4)_{10} = 2$ (ост. 3) $_{10} = 2$ (ост. 3) $_{16}$;
 $34_{16} = (3 \cdot 16^1 + 4)_{10} = 52_{10}$;
 $(52:4)_{10} = 13_{10} = D_{16}$.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Деление и умножение в произвольной системе счисления производятся по тем же правилам, что и в десятичной системе.

ПРИМЕНЕНИЕ ЗНАНИЙ

Вычислите значения следующих выражений:

а) $234,5_8 + 15,2_8 \cdot 21_8 - 12_8$;

б) $1110011_2 : 11001_2$;

в) $271_{16} : 19_{16}$;

г) $255_8 + 11001,11_2 \cdot (60_8 + 14_{10}) - 1F_{16}$. Ответ запишите в четверичной системе счисления.

§ 7. Запись числа в общем виде

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Теперь вы знаете, как раскладывается число по степеням основания в любой позиционной системе счисления.

Проанализируйте, что общего во всех таких записях и чем они различаются. Вспомните, как с развёрнутой формой связано обычное представление числа.

■ Какие закономерности вы обнаружили? Сформулируйте проблему урока. Сравните свой вариант с авторским (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните основные характеристики системы счисления (основание и базис). (§ 1)

Вспомните, что такое развёрнутая форма записи числа. (§ 1)

Вспомните, как переводится в десятичную систему счисления число, записанное в произвольной системе счисления. (§ 2)

РЕШЕНИЕ ПРОБЛЕМЫ

Давайте вспомним, как переводится в десятичную систему счисления число, записанное в произвольной системе счисления. Нас не будет в данный момент интересовать результат перевода (не надо выполнять арифметические действия), обратим внимание на запись этих чисел.

Примеры:

$$100,122_5 = 1 \cdot 5^2 + 0 \cdot 5^1 + 0 \cdot 5^0 + 1 \cdot 5^{-1} + 2 \cdot 5^{-2} + 2 \cdot 5^{-3};$$

$$11010,110_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}.$$

Выведем теперь общий вид записей чисел в развёрнутой форме.

1. Обозначим основание системы счисления буквой p , степень основания буквой m , а коэффициент при p^m обозначим буквой a с индексом m (равным степени основания).
2. Убывающие на 1 в каждом следующем слагаемом степени основания будут обозначены как m , $m-1$, $m-2$ и т. д.
3. Основания в этих степенях будут выглядеть как p^m , p^{m-1} , p^{m-2} и т. д.
4. Коэффициенты при этих слагаемых будут обозначены соответственно a_m , a_{m-1} , a_{m-2} и т. д.

Число в любой позиционной системе счисления можно представить в **общем виде**:

$$N = \underbrace{a_m \cdot p^m + a_{m-1} \cdot p^{m-1} + \dots + a_0 \cdot p^0}_{\text{целая часть}} + \underbrace{a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + \dots + a_{-k} \cdot p^{-k}}_{\text{дробная часть}},$$

где m – максимальная для этого числа положительная степень основания, k – максимальная отрицательная ($-k$ – минимальная!) степень основания. От m до 0 – область целой части, от -1 до $-k$ – дробной.

Пример 1

Число $N = 893,705_{10} = 8 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0 + 7 \cdot 10^{-1} + 0 \cdot 10^{-2} + 5 \cdot 10^{-3}$ представляется в общем виде так:

$$N = a_2 p^2 + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + a_{-2} p^{-2} + a_{-3} p^{-3},$$

$$\text{где } p = 10, a_2 = 8, a_1 = 9, a_0 = 3, a_{-1} = 7, a_{-2} = 0, a_{-3} = 5.$$

Наиболее часто число в общем виде представляется только в виде совокупности коэффициентов, записанных в соответствующих позициях. Разделителем целой и дробной части служит запятая:

$$N = \underbrace{a_m a_{m-1} \dots a_0}_{\text{целая часть}}, \underbrace{a_{-1} a_{-2} \dots a_{-k}}_{\text{дробная часть}}$$

Пример 2

Число $N = 321,02_4$ представим в развёрнутой форме:

$$321,02_4 = 3 \cdot 4^2 + 2 \cdot 4^1 + 1 \cdot 4^0 + 0 \cdot 4^{-1} + 2 \cdot 4^{-2}.$$

Общий вид этого числа в виде совокупности коэффициентов:

$$N = a_2 a_1 a_0, a_{-1} a_{-2},$$

$$\text{где } a_2 = 3, a_1 = 2, a_0 = 1, a_{-1} = 0, a_{-2} = 2.$$

То есть запись 321,02 – это и есть представление числа в виде совокупности коэффициентов.

Замечание. Обратите внимание: если в представлении числа в развёрнутой форме отсутствует какая-то степень, то, с учётом того, что система счисления – позиционная, в месте, соответствующем этой степени, в записи числа в виде совокупности коэффициентов должен появиться 0. Не забывайте про это.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Для всех позиционных систем счисления существует общий вид числа:

$$N = a_m \cdot p^m + a_{m-1} \cdot p^{m-1} + \dots + a_0 \cdot p^0 + a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + \dots + a_{-k} \cdot p^{-k},$$

где p – основание системы счисления, m – максимальная для этого числа положительная степень основания, k – максимальная отрицательная ($-k$ – минимальная!) степень основания, a_m, \dots, a_{-k} – коэффициенты при степенях основания.

Наиболее часто число представляется только в виде совокупности коэффициентов, записанных в соответствующих позициях:

$$N = a_m a_{m-1} \dots a_0 a_{-1} a_{-2} \dots a_{-k}.$$

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Запишите число в развёрнутой форме. Укажите, чему равно основание p и коэффициенты a_m, \dots, a_{-k} при соответствующих степенях основания:

а) $2376,26_8$;

б) $304,01_6$;

в) $10A,23_{16}$.

2. Запишите число в виде совокупности коэффициентов, записанных в соответствующих позициях:

а) $2 \cdot 5^3 + 3 \cdot 5^1 + 1 \cdot 5^{-1}$;

б) $1 \cdot 16^4 + 3 \cdot 16^2 + 0 \cdot 16^{-2}$;

в) $1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^0 + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$.

Подсказка. Для решения следующих задач запишите числа в развёрнутой форме, обозначив неизвестное основание системы счисления как p .

3. Учитель записал, что в олимпиаде по информатике принимало участие 10 учеников 8-го класса и 50 учеников 9-го класса. Общее количество учеников – 100. В какой системе счисления учитель записал данные?

4. 60 груш разрезали пополам, и получилось 150 половинок. В какой системе счисления записаны числа? Чему равно количество груш в десятичной системе счисления?

§ 8. Кодирование чисел. Представление чисел (беззнаковых и целых) в памяти компьютера

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

До сих пор вы изучали числа, представленные в разных системах счисления, в том числе и двоичной, без связи с компьютером. Множество чисел (как целых, так и вещественных) бесконечно. Никогда при «ручных» вычислениях не возникает проблемы, что число куда-то «не поместится». Другое дело – компьютер. Одна из его важных характеристик – объем оперативной памяти. Она не бесконечна и определяется характеристиками компьютера. Числа разных типов требуют для своего размещения разное количество памяти.

Сегодня мы говорим о целых числах. Есть ли разница в том, как мы записываем числа, и в том, как они представлены в памяти компьютера?

- Какую проблему вы видите? Сформулируйте тему урока. Сравните свою формулировку с авторской (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните, как переводятся числа из десятичной системы счисления в двоичную и обратно. (§ 2)

Вспомните, как переводятся двоичные числа в восьмеричную и шестнадцатеричную системы счисления. (§ 3)

РЕШЕНИЕ ПРОБЛЕМЫ

Познакомимся с некоторыми важными терминами.

Разряд – элемент памяти, способный находиться в одном из двух состояний (0/1), служащий для хранения одного разряда двоичного числа. Таким образом, в одном разряде хранится 1 бит данных.

Ячейка памяти – минимальная адресуемая часть памяти компьютера. Она может составлять для конкретной машины 1 байт, 2 байта и т. п.

Существуют два основных формата представления чисел в памяти компьютера. Один из них используется для кодирования целых чисел – это формат с **фиксированной запятой** (говорят также: с фиксированной точкой). В этом случае каждому разряду памяти соответствует всегда один и тот же разряд числа, а запятая «располагается» справа от младшего разряда (вне разрядной сетки). Второй формат служит для представления чисел с **плавающей запятой** (точкой), используется для задания вещественных (дробных) чисел. Его мы рассмотрим в следующем параграфе.

Для размещения чисел разных типов используется разное количество ячеек (байтов) памяти.

Обратим внимание на представление целых чисел в памяти компьютера.

ЦЕЛЫЕ ЧИСЛА БЕЗ ЗНАКА

Для представления целых чисел без знака может использоваться 1 байт или 2 байта. Максимально возможное число без знака:

Однобайтовый формат (8 битов):

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

 $2^8 - 1 = 255$

Двухбайтовый формат (16 битов):

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $2^{16} - 1 = 65\,535$

ЦЕЛЫЕ ЧИСЛА СО ЗНАКОМ

Для представления целых чисел со знаком в компьютере используются прямой, обратный и дополнительный коды числа.

Для **положительного числа** прямой, обратный и дополнительный коды совпадают и представляют двоичный код числа. Старший (крайний левый) бит в представлении числа отводится под знак числа и для положительного числа содержит 0.

Отрицательное число в прямом, обратном и дополнительном кодах отображается по-разному:

- **прямой код** — двоичный код модуля числа;
- **обратный код** — инвертированный (включая знак числа) двоичный код модуля числа. Инвертировать бит числа — значит изменить его значение на противоположное: 0 на 1, а 1 на 0 (вспомните, в § 4 мы уже выполняли инвертирование двоичного числа);
- **дополнительный код** получается путём прибавления 1 к обратному коду.

Отрицательные целые числа в компьютере представляются в дополнительном коде. В знаковом разряде при этом находится 1 — признак отрицательного числа.

Алгоритм получения дополнительного кода отрицательного числа:

1. Найти двоичный код модуля этого числа.
2. Инвертировать содержимое всех разрядов, включая разряд знака.
3. Прибавить 1 к обратному коду.

Пример

Представление в обратном и дополнительном коде числа -67_{10} :

Степени основания ($p = 2$):

1. Прямой код модуля числа $|-67|$:

2. Обратный код числа -67 :

7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1
1	0	1	1	1	1	0	0
							1
1	0	1	1	1	1	0	1

3. Дополнительный код числа -67 :

Отрицательные числа при вводе в машину преобразуются в дополнительный код и в таком виде хранятся, участвуют в операциях, перемещаются, а **при выводе результата происходит обратное преобразование** в отрицательное десятичное число. Для этого дополнительный код инвертируется, и к результату прибавляется 1. Получается модуль числа, который и переводится в десятичную систему счисления. Потом добавляем знак «-» – и готово.

Диапазон значений целых чисел со знаком на примере однобайтового представления:

Максимальное число							
7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1
127							

Минимальное число							
7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0
-128							

Целые числа со знаком обычно занимают в памяти компьютера 1, 2 или 4 байта. Диапазоны значений целых чисел со знаком:

Формат числа, байты	Значение
1	-128...127
2	-32 768...32 767
4	-2 147 483 648...2 147 483 647

Задача. Число $N = -67$ представлено в однобайтовом формате в дополнительном коде. Затем чётные биты были инвертированы (0-й бит считать чётным). Какому десятичному числу соответствует полученный код? Какому шестнадцатеричному числу соответствует полученный код?

Решение

1. Находим двоичное выражение $|-67|$.
2. Находим обратный код.
3. Вычисляем дополнительный код, получаем код числа -67 .
4. Инвертируем содержимое чётных разрядов.

7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1
1	0	1	1	1	1	0	0
1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	0

Определяем модуль полученного отрицательного числа:

1. Инвертируем содержимое всех разрядов.
2. Прибавляем 1.
3. Получаем модуль шестнадцатеричного числа: 18_{16} .
4. Получаем модуль десятичного числа: 24_{10} .

0	0	0	1	0	1	1	1
0	0	0	1	1	0	0	0
1				8			
$1 \cdot 2^4 + 1 \cdot 2^3 = 24_{10}$							

Ответ: десятичное число: -24_{10} ; шестнадцатеричное число: -18_{16} .

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Числа в компьютере кодируются двоичным кодом. Целые числа представляются в формате с фиксированной запятой. Целые положительные числа представляются в прямом коде (это двоичный код числа), в старший бит записывается 0. Для представления целого отрицательного числа служат прямой, обратный и дополнительный коды. Отрицательные числа в памяти компьютера представляются в дополнительном коде. Старший бит целого отрицательного числа содержит 1. При выводе результата происходит обратное преобразование: дополнительный код инвертируется, и к нему прибавляется 1.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Запишите 8-битовое представление в памяти компьютера следующих чисел: -25_{10} , 117_{10} , 25_{10} . Как будут выглядеть шестнадцатеричные коды этих чисел?
2. Запишите 12-битовое представление в памяти компьютера следующих чисел: 123_{10} , -23_{10} . Как будут выглядеть шестнадцатеричные коды этих чисел?
3. Для хранения целого числа -89_{10} в дополнительном коде используется однобайтовое представление. Сколько единиц оно содержит?

§ 9. Запись числа в нормализованном виде. Числа с плавающей запятой. Представление вещественных чисел в памяти компьютера

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Вы имеете большой опыт вычислений, в которых используются дробные (вещественные) числа. Очень большие и очень маленькие числа участвуют в вычислениях при решении задач по физике и химии. Оперировать с такими числами в виде, как мы их привыкли записывать, неудобно. Вы узнаете, как записать такие числа, чтобы вычисления стали более эффективными.

- Как вы считаете, как решается указанная проблема? Сформулируйте тему урока. Сравните свой вариант с авторским (с. 142 учебника).

РЕШЕНИЕ ПРОБЛЕМЫ

СТАНДАРТНЫЙ ВИД ЧИСЛА

Для записи больших чисел, например данных о Земле: радиус – R_z (средний) = 6 371 000 м; масса – $M_z = 6\,000\,000\,000\,000$ миллиардов тонн; или очень маленьких, например данных об электроны: масса – $M_e = 0,0000...09$ кг (30 нулей после запятой!), неудобно пользоваться числами, записанными в привычном виде. Такие числа записываются в **стандартном виде** $N = \pm a \cdot 10^q$, где $0 < a < 10$, q – **порядок** числа (степень основания системы счисления), a – **мантисса** числа:

R_z (средний) = 6 371 000 м = $6,371 \cdot 10^6$ м, $q = 6$ (порядок числа равен 6);

$M_z = 6\,000\,000\,000\,000$ млрд тонн = $6 \cdot 10^{12}$ млрд тонн = $6 \cdot 10^{24}$ кг, $q = 24$;

$M_e = 0,0...09$ кг = $9 \cdot 10^{-31}$ кг, $q = -31$ (для дробных чисел порядок отрицательный).

Как видите, **знак порядка** будет положительным для чисел, больших 1, и отрицательным для чисел, меньших 1. Насколько компактной получается запись!

Кроме удобства записи, у чисел, записанных в стандартном виде, есть самое главное преимущество: действия над ними производить гораздо легче, и меньше возможность ошибиться.

НОРМАЛИЗОВАННЫЙ ВИД ЧИСЛА

Число в нормализованном виде выглядит так:

$N = \pm m \cdot p^q$, где m – **мантисса** числа, q – **порядок** числа, p – основание системы счисления. Мантисса должна быть меньше 1, и первая её значащая цифра не должна быть нулём. (Вспомним, что для чисел, меньших 1, $q < 0$; например: $0,0023_{10} = 0,23 \cdot 10^{-2}_{10}$.)

При записи числа в нормализованной форме принято мантиссу и порядок записывать в соответствующей системе счисления, а основание системы счисления – в десятичной системе.

Примеры

Числа в нормализованной форме:

$$3,5_{10} = 0,35 \cdot 10^1_{10}; \quad 1101,011_2 = 0,1101011 \cdot 2^{100}_2 \quad (m = 0,1101011_2; \\ p = 2_{10}; \quad q = 4_{10} = 100_2)$$

ПРЕДСТАВЛЕНИЕ ВЕЩЕСТВЕННЫХ ЧИСЕЛ В КОМПЬЮТЕРЕ

Для представления вещественных чисел в компьютере используется запись числа **с плавающей запятой** (говорят также: с плавающей точкой). Предварительно вещественное число (так же, как и целое) должно быть переведено в двоичную систему счисления.

При представлении чисел с плавающей запятой в старший бит записывается знак числа, часть разрядов отводится для записи порядка числа, остальные – для записи значащей части. Для того чтобы не хранить знак порядка (он может быть положительный или отрицательный), используется способ представления чисел с помощью смещённого порядка (СП). К истинному значению порядка прибавляется смещение. Смещение выбирается так, чтобы минимальному значению порядка соответствовал 0.

Смещённый порядок рассчитывается по формуле:

$$\text{СП} = (2^{k-1} - 1) + q,$$

где q – **истинный порядок** числа, k – количество разрядов, отводимых в ячейке для записи смещённого порядка.

В зависимости от представления числа (от типа числа) под порядок отводится различное количество разрядов. Например, в 4-байтовом представлении для записи порядка отводится 1 байт (8 битов), $k = 8$. $2^{k-1} - 1 = 2^{8-1} - 1 = 127_{10}$. Таким образом, в четырехбайтовом представлении может разместиться число, имеющее порядок $[q]$, принимающий значения в диапазоне от -128 до $+127$. Формула для СП будет выглядеть так: $\text{СП} = 1111111_2 + q_2$. Значения смещённого порядка (СП) при этом меняются от 0 до 255_{10} .

Существует стандарт для представления чисел с одинарной точностью (4 байта) и с двойной точностью (8 байтов). Он основан на том факте, что для представления числа в памяти компьютера его записывают таким образом, что в значащей части, перед запятой всегда 1. В целях экономии ресурсов были установлены следующие правила: 1) для хранения этой единицы не выделяется дополнительный бит: единица подразумевается, как и двоичная запятая; 2) в данном стандарте принято, что мантисса всегда больше 1; то есть диапазон значащей части лежит в пределах от 1 до 2.

Алгоритм представления числа с плавающей запятой:

1. Перевести модуль числа из системы счисления с основанием p в двоичную систему счисления.
2. Представить двоичное число в нормализованной форме $\{1 \leq m < 2\}$.
3. Вычислить смещённый порядок числа.
4. Разместить знак числа, СП и мантиссу в соответствующих разрядах ячейки.

Задача 1

Требуется представить число -147.125_{10} в компьютерном виде, в 4-байтовом представлении. Найти его шестнадцатеричный код.

Решение

1. Найдём двоичное представление числа $|-147,125|$:

a) $147_{10} = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0 = 10010011_2;$

6) $0,125_{10} = 0,001_2$.

$$147,125 = 10010011,001_2$$

2. Представим это число в нормализованной форме:

$$10010011,001 = 1,0010011001 \cdot 2^{11}; q=111; m=1,0010011001_p.$$

3. $СП = 1111111_2 + 111_2 = 10000110_2$.

4. Разместим данные в памяти, убирая 1 из значения мантиссы и помня, что старший бит отводится под знак числа. Так как в данном случае число отрицательное, пишем 1.

[illegible]

* — знак числа (мантиссы): 0 — для положительного числа, 1 — для отрицательного.

Шестнадцатеричный код занимает меньше места: C31324000:

1 1 0 0	0 0 1 1	0 0 0 1	0 0 1 1	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0
C	3	1	3	2	0	0	0

Задача 2

Значение переменной представлено в формате с плавающей запятой и выражено в шестнадцатеричной системе счисления: $N = C2530000_{16}$. Требуется найти десятичное значение числа.

Решение

1. Найдём двоичное выражение шестнадцатеричного числа:

1 1 0 0	0 0 1 0	0 1 0 1	0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
C	2	5	3	0	0	0	0

2. В старшем бите записана 1, значит, число отрицательное.

3. Выделим смещённый порядок (отсчитаем 8 позиций от старшего бита):
 $СП = 10000100$.

4. Найдём истинный порядок (q) .

Так как $СП = (2^{8-1} - 1) + q$, то $q = 10000100_2 - 1111111_2 = 101_2$, $q = 5_{10}$.

5. Учитывая «убранную» 1, запишем мантиссу: $m = 1,1010011$.

6. Учитывая порядок $(q = 5_{10})$, запишем двоичное число:

$$N = -110100,11_2$$

7. Находим десятичное число: $N = -52,75_{10}$.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Существуют различные способы записи вещественных (дробных) чисел.

При вычислении выражений с очень большими и очень маленькими десятичными числами «вручную» удобно пользоваться числами, записанными в стандартном виде: $N = \pm a \cdot 10^q$, где $0 < a < 10$, $q < 0$ для $a < 1$, $q > 0$ для $a > 1$.

Число в нормализованном виде выглядит так: $N = \pm m \cdot p^q$, причём мантисса ($m < 1$, первая цифра мантиссы больше 0) и порядок (q) записываются в текущей системе счисления, а основание системы счисления (p) – в десятичной.

Для представления вещественных чисел в компьютере используется запись числа с плавающей запятой (точкой). Предварительно вещественное число (так же, как и целое) должно быть переведено в двоичную систему счисления и представлено в нормализованной форме ($1 < m < 2$).

Для того чтобы не хранить знак порядка (он может быть положительный или отрицательный), используется способ представления чисел с помощью смещённого порядка.

ПРИМЕНЕНИЕ ЗНАНИЙ

1. Объясните, чем отличается запись числа в стандартном виде от записи в нормализованном виде.

2. Как изменяются правила записи числа в нормализованном виде для представления данного числа в компьютере?

3. Запишите числа в стандартном виде.

а) $-124\,834,34_{10}$;

б) $0,00004040045_{10}$;

в) 6000000000000_{10} .

4. Запишите числа в нормализованном виде.

а) $-32423,22_5$;

б) $11100112,22_3$;

в) $0,00000111_2$.

5. Как представлены в компьютере в четырёх байтах следующие числа? Определите их шестнадцатеричные коды.

а) $-25,25_{10}$;

б) $188,125_{10}$.

§ 10. Сложение целых чисел в памяти компьютера

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Вы знаете, что числа в компьютере кодируются двоичным кодом, и знакомы с некоторыми способами их размещения в памяти компьютера. Но в оперативной памяти числа не только хранятся, из ячеек памяти процессор берёт данные для вычислений. Вы умеете выполнять арифметические действия с двоичными числами «вручную», но в то же время знаете, что представление чисел в компьютере отличается от того, в каком виде вы их используете для вычислений.

• Как вы считаете, какая проблема в этой ситуации? Сформулируйте тему урока. Сравните свой вариант с авторским (с. 142 учебника).

НЕОБХОДИМЫЕ БАЗОВЫЕ ЗНАНИЯ

Вспомните, как представляются целые числа в памяти компьютера. (§ 8)

РЕШЕНИЕ ПРОБЛЕМЫ

Мы будем говорить о сложении целых чисел в компьютере. Почему из всех арифметических действий мы выбрали именно сложение? Дело в том, что одной операции сложения достаточно для того, чтобы через неё получить все другие.

Разберём алгоритм сложения на примерах, рассмотрим все варианты знаков чисел.

Найдём для дальнейшего использования прямой, обратный и дополнительный коды целых чисел A и B , где $|A| = 8$, $|B| = 7$:

Код	$ A = 8$								$ B = 7$							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Прямой	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
Обратный	1	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0
Дополнительный	1	1	1	1	1	0	0	0	1	1	1	1	1	0	0	1

Алгоритм сложения чисел в компьютере:

1. Разместить слагаемые в разрядных сетках в прямых кодах.
2. Отрицательное слагаемое (слагаемые) преобразовать в дополнительный код.
3. Слагаемые складываются по правилам сложения двоичных чисел. При этом знаковые разряды участвуют в вычислениях наряду с числовыми разрядами.
4. Единицу переноса из знакового разряда (если таковая возникнет) отбросить.

5. Если результат положителен, то он представлен в прямом коде и не требует никаких преобразований. Если результат отрицателен, то он представлен в дополнительном коде. По известным правилам преобразовать его в прямой код модуля числа. Знак «-» дописать к результату.

• Вспомните правила преобразования чисел из дополнительного кода в прямой. Что при этом получается?

Разберём возможные случаи соотношения знаков слагаемых A и B .

1. Оба числа положительные ($A = 8, B = 7$):

Степени двойки	7	6	5	4	3	2	1	0
A (прямой код)	0	0	0	0	1	0	0	0
B (прямой код)	0	0	0	0	0	1	1	1
Сумма (прямой код), единица переноса не возникает	0	0	0	0	1	1	1	1
Ответ: $A + B = 1111_2 = 15_{10}$	0	0	0	0	1	1	1	1

2. Оба числа отрицательные ($A = -8, B = -7$):

Степени двойки	7	6	5	4	3	2	1	0
A (дополнительный код)	1	1	1	1	1	0	0	0
B (дополнительный код)	1	1	1	1	1	0	0	1
Сумма (дополнительный код), единица переноса отбрасывается	1	1	1	1	0	0	0	1
Получение модуля числа. Ответ: $A + B = -1111_2 = -15_{10}$	0	0	0	0	1	1	1	1

3. Числа разных знаков, положительное число по модулю больше отрицательного ($A = 8, B = -7$):

Степени двойки	7	6	5	4	3	2	1	0
A (прямой код)	0	0	0	0	1	0	0	0
B (дополнительный код)	1	1	1	1	1	0	0	1
Сумма (прямой код), единица переноса отбрасывается	0	0	0	0	0	0	0	1
Ответ: $A + B = 1_2 = 1_{10}$	0	0	0	0	0	0	0	1

4. Числа разных знаков, положительное число по модулю меньше отрицательного ($A = -8, B = 7$):

Степени двойки	7	6	5	4	3	2	1	0
A (дополнительный код)	1	1	1	1	1	0	0	0
B (прямой код)	0	0	0	0	0	1	1	1
Сумма (дополнительный код), единица переноса не возникает	1	1	1	1	1	1	1	1
Получение модуля числа. Ответ: $A + B = -1_2 = -1_{10}$	0	0	0	0	0	0	0	1

• Самостоятельно найдите прямой и дополнительный коды чисел 87 и 67.

5. Оба числа положительные, их сумма не меньше, чем $2n-1$ ($A = 87, B = 67, A + B \geq 128$, для восьмибитового представления):

Степени двойки	7	6	5	4	3	2	1	0
A (прямой код)	0	1	0	1	0	1	1	1
B (прямой код)	0	1	0	0	0	0	1	1
Сумма – несовпадение знака суммы со знаками слагаемых	1	0	0	1	1	0	1	0
Ошибка: появилась единица в знаковом разряде.								

6. Оба числа отрицательные, сумма их модулей не меньше, чем $2n-1$ ($A = -87, B = -67, |A| + |B| \geq 128$, для восьмибитового представления):

Степени двойки	7	6	5	4	3	2	1	0
A (дополнительный код)	1	0	1	0	1	0	0	1
B (дополнительный код)	1	0	1	1	1	1	0	1
Сумма – несовпадение знака суммы со знаками слагаемых	0	1	1	0	0	1	1	0
Ошибка: переполнение разрядной сетки								

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Суммирование целых чисел происходит поразрядно.

Отрицательное число представляется в дополнительном коде.

Единица переноса (если она возникает) отбрасывается.

Если результат положителен, он представлен в прямом коде и не требует никаких преобразований. Если результат отрицателен, то он представлен в дополнительном коде. По известным правилам он преобразуется в прямой код модуля числа.

ПРИМЕНЕНИЕ ЗНАНИЙ

Выполните сложение десятичных чисел. Укажите, в каких случаях имеет место переполнение разрядной сетки.

а) $100 - 56$;

б) $-110 + 66$;

в) $125 + 100$;

г) $-45 - 34$.

Проверь себя**Задание 1**

Найдите произведение чисел 45_8 и 12_8 .

Задание 2

Найдите частное от деления 10000100_2 на 1100_2 .

Задание 3

Как представлено десятичное число 117 в памяти компьютера?

Задание 4

Как представлено десятичное число -234 в памяти компьютера? Найдите его шестнадцатеричный код для 12-битового представления.

Задание 5

Запишите десятичное число 25,5 в 4-байтовом представлении. Каков его шестнадцатеричный код?

ПРИМЕНЯЕМ ЗНАНИЯ (необходимый уровень)

Задание 1

Определите число по приведённым данным – основанию системы счисления и коэффициентам в его записи в общей форме:

$$p = 4, a_3 = 1, a_2 = 0, a_1 = 3, a_0 = 0, a_{-1} = 1.$$

Задание 2

Представьте целые десятичные числа в однобайтовом формате.

- а) 115;
- б) -100;
- в) 122;
- г) 65.

Задание 3

Представьте целые десятичные числа в однобайтовом формате. Определите шестнадцатеричный код представления.

- а) -117;
- б) -88;
- в) -100;
- г) -87.

Задание 4

Найдите произведение чисел.

- а) 123_4 и 111_4 ;
- б) 110111_2 и 10111_2 ;
- в) 1022_3 и 222_3 .

Задание 5

Найдите частное от деления чисел.

- а) 101001111_2 и 1000011_2 ;
- б) 1111101_2 и 11001_2 ;
- в) 100110100_2 и 10110_2 .

Задание 6

Десятичное число $N = -100$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 8 двоичных разрядов. Определите, какому восьмеричному числу соответствует код.

Задание 7

Как представляется в памяти компьютера десятичное число $-112,25$?

ПРИМЕНЯЕМ ЗНАНИЯ (повышенный уровень)**Задание 1**

Числа записаны в дополнительном коде в восьмибитовом представлении. Определите, какому шестнадцатеричному числу будет соответствовать этот код.

- а) 10011111;
- б) 11001010;
- в) 11101010;
- г) 11000001.

Задание 2

Числа записаны в дополнительном коде в восьмибитовом представлении. Найдите их десятичные представления.

- а) 10000111;
- б) 01111010;
- в) 00101010;
- г) 11011101.

Задание 3

Как представлены вещественные числа в компьютере (в четырёхбайтовом представлении)?

- а) $-123,23_{10}$;
- б) $1011011010,11101_2$.

Задание 4

Выполните операцию $A + B$. Отрицательное число представьте в дополнительном коде. Формат – 1 байт.

- а) $A = 9, B = -2$;
- б) $A = -17, B = 12$.

Задание 5

Десятичное число $N = -117$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 12 двоичных разрядов. Определите, какому шестнадцатеричному числу соответствует код.

Задание 6

Найдите произведение восьмеричных чисел 317_8 и 25_8 .

Задание 7

Найдите частное от деления четверичных чисел 3320_4 и 20_4 .

ПРИМЕНЯЕМ ЗНАНИЯ (максимальный уровень)
Задание 1

Шестизначное десятичное число оканчивается цифрой 4. Если эту цифру переставить из конца числа в начало, то есть приписать её перед первой цифрой, не изменяя порядок остальных пяти, то получится число, которое в четыре раза больше первоначального. Найдите это число.

Подсказка. Представим первое число: $A = 10 \cdot y + 4$. (Что обозначим как y ?) Тогда второе число: $B = 4 \cdot 10^n + y$. (Что обозначим как n ? Чему равно n для данной задачи?) Составив и решив уравнение, получите ответ.

Задание 2

Десятичное число $N = -145$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 16 двоичных разрядов. Определите, какому шестнадцатеричному числу будет соответствовать код, если значения битов с номерами 0, 1, 2, 3 и 4 инвертировать. Номера битов отсчитываются справа налево, начиная с нуля. Младший бит – бит с номером 0.

Задание 3

Даны два числа: десятичное число $A = -30_{10}$ и шестнадцатеричное число $B = -20_{16}$. В памяти компьютера эти числа представлены в формате с фиксированной запятой в дополнительном коде. Длина формата – 12 двоичных разрядов. Выполните операцию $A + B$ – определите восьмеричный дополнительный код результата операции.

Задание 4

Выполните операцию $A + B$, где $A = 2$, $B = -9$. Отрицательное число представлено в дополнительном коде. Формат – 1 байт. Представьте код результата в шестнадцатеричной системе счисления.

Задание 5

Найдите произведение чисел.

а) 1011_{16} и 123_{16} ;

б) 227_8 и 17_8 .

Задание 6

Найдите частное от деления чисел $E65_{16}$ и 37_{16} .

Итоговая проверочная работа

Задание 1

- а) Десятичное число $N = -89$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 8 двоичных разрядов. Определите этот код.
- б) Найдите произведение чисел 1101101_2 и 1101_2 .
- в) Найдите сумму чисел $A = 12$, $B = 6$ в восьмибитовом формате. Ответ запишите в шестнадцатеричном коде.
- г) Запишите десятичную дробь $133,125$ в представлении в памяти компьютера.

Задание 2

- а) Десятичное число $N = -121$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 12 двоичных разрядов. Определите шестнадцатеричный код этого представления.
- б) Найдите произведение чисел $24,5_8$ и 47_8 .
- в) Найдите сумму чисел $A = -12$, $B = -15$ в восьмибитовом формате. Код суммы запишите в восьмеричной системе счисления.
- г) Запишите десятичную дробь $-145,25$ в представлении в памяти компьютера.

Задание 3

- а) Десятичное число $N = -151$ представлено в формате с фиксированной запятой в дополнительном коде. Длина формата – 12 двоичных разрядов. Определите, какому десятичному числу будет соответствовать код, если значения битов с чётными номерами (бит с номером 0 считать чётным) инвертировать. Номера битов отсчитываются справа налево, начиная с нуля. Младший бит – бит с номером 0.
- б) Трёхзначное десятичное число оканчивается цифрой 3. Если эту цифру переместить на два разряда влево, то есть так, что с неё будет начинаться запись нового числа, то это новое число будет на единицу больше утроенного исходного числа. Найдите исходное число.
- в) Найдите сумму чисел $A = -112$, $B = -126$ в восьмибитовом формате. Ответ запишите в шестнадцатеричном коде.
- г) Найдите произведение чисел $24,7_8$ и $22,2_8$. Ответ запишите в шестнадцатеричной системе счисления.
- д) Найдите частное от деления EE_{16} и 70_{16} . Ответ запишите в двоичной системе счисления.

Содержание

Дорогие ребята!.....	3
Модуль 1. Алгоритмизация и программирование	9
Введение	10
§ 1. Знакомство с математической логикой	11
<i>Что означают понятия «логика» и «математическая логика»?</i>	
§ 2. Поиск в массиве.....	20
<i>Как искать данные в массиве?</i>	
§ 3. Упорядочение массивов	25
<i>Что такое упорядочение массивов и как оно происходит?</i>	
§ 4. Структурирование программ. Подпрограммы	33
<i>Что делать, если алгоритм включает в себя повторяющийся несколько раз другой алгоритм?</i>	
§ 5. Передача параметров в подпрограммы	40
<i>Как данные передаются в подпрограммы и каким образом подпрограммы передают основной программе результаты своей работы?</i>	
§ 6. Знакомство с математической логикой: продолжение	52
<i>Какие возможности предоставляет математическая логика?</i>	
§ 7. Использование констант и собственных типов	59
<i>Как сделать написание и чтение программ более удобным?</i>	
§ 8. Работа с упорядоченными массивами	64
<i>Какими преимуществами обладает упорядоченный массив по сравнению с неупорядоченным?</i>	
§ 9–10. Поговорим об эффективности	70
<i>Что такое эффективная программа?</i>	
Модуль 2. Системы счисления	85
Введение	86
§ 1. Что такое системы счисления	87
<i>От чего зависят правила записи чисел?</i>	
§ 2. Перевод числа из произвольной системы счисления в десятичную. Перевод целого числа из десятичной системы счисления в произвольную.....	92
<i>Существуют ли правила преобразования чисел из одной системы счисления в другую? Каковы они?</i>	
§ 3. Переход между системами счисления, основания которых – степени двойки.....	95
<i>Существует ли другой способ перевода чисел из одной системы счисления в другую, если у них основания – степень двойки?</i>	

§ 4. Сложение и вычитание чисел в произвольных системах счисления	100
<i>Как производить арифметические действия в произвольной позиционной системе счисления?</i>	
§ 5. Перевод правильной десятичной дроби в произвольную систему счисления	107
<i>Как перевести из десятичной системы счисления в произвольную десятичную дробь?</i>	
§ 6. Деление и умножение в позиционных системах счисления	115
<i>Как делить и умножать в произвольной системе счисления?</i>	
§ 7. Запись числа в общем виде	119
<i>Можно ли найти общий вид записи чисел в позиционной системе счисления?</i>	
§ 8. Кодирование чисел. Представление чисел (беззнаковых и целых) в памяти компьютера	122
<i>Как представлены целые числа в компьютере?</i>	
§ 9. Запись числа в нормализованном виде. Числа с плавающей запятой. Представление вещественных чисел в памяти компьютера.....	126
<i>Как представлены вещественные числа в компьютере?</i>	
§ 10. Сложение целых чисел в памяти компьютера	131
<i>Как выполняются арифметические действия в компьютере?</i>	

Горячев, А.В.

Информатика. 8 кл. : учеб. для общеобразоват. учреждений : в 2-х кн. Кн. 2 / А.В. Горячев, В.Г. Герасимова, Л.А. Макарина, А.В. Паволоцкий, А.А. Семёнов, Т.Л. Чернышёва. – М. : Баласс, 2013. – 144 с. : ил. (Образовательная система «Школа 2100»).

ISBN 978-5-85939-940-6 (кн. 2)

ISBN 978-5-85939-997-0

Учебник предназначен для учащихся 8-го класса образовательных учреждений. Соответствует Федеральному государственному образовательному стандарту основного общего образования, является продолжением непрерывного курса информатики и составной частью комплекта учебников развивающей Образовательной системы «Школа 2100».

Содержание учебника представлено в виде отдельных учебных модулей, из которых учитель может выбрать нужные в соответствии с требованиями основной образовательной программы школы. Учебный материал предлагается на необходимом и повышенном уровне.

УДК 373.167.1:004+004(075.3)
ББК 32.81я721

**Горячев Александр Владимирович, Герасимова Вера Георгиевна,
Макарина Любовь Александровна, Паволоцкий Александр Владимирович,
Семёнов Андрей Александрович, Чернышёва Татьяна Леонидовна**

Информатика
Учебник для 8-го класса
Книга 2

Концепция оформления
и художественное редактирование – *Е.Д. Ковалевская*

Подписано в печать 30.07.12. Формат 70х90/16. Гарнитура Европа.
Печать офсетная. Бумага офсетная. Объем 9 п.л. Тираж 4 000 экз. Заказ № 32556 (Sm - Sm).

Общероссийский классификатор продукции ОК-005-93, том 2; 953005 – литература учебная

Издательство «Баласс». 109147 Москва, Марксистская ул., д. 5, стр. 1

Почтовый адрес: 111123 Москва, а/я 2, «Баласс»

Телефоны для справок: (495) 368-70-54, 672-23-12, 672-23-34

<http://www.school2100.ru> E-mail: balass.izd@mtu-net.ru

Отпечатано в ОАО «Смоленский полиграфический комбинат»
214020 Смоленск, ул. Смольянинова, 1