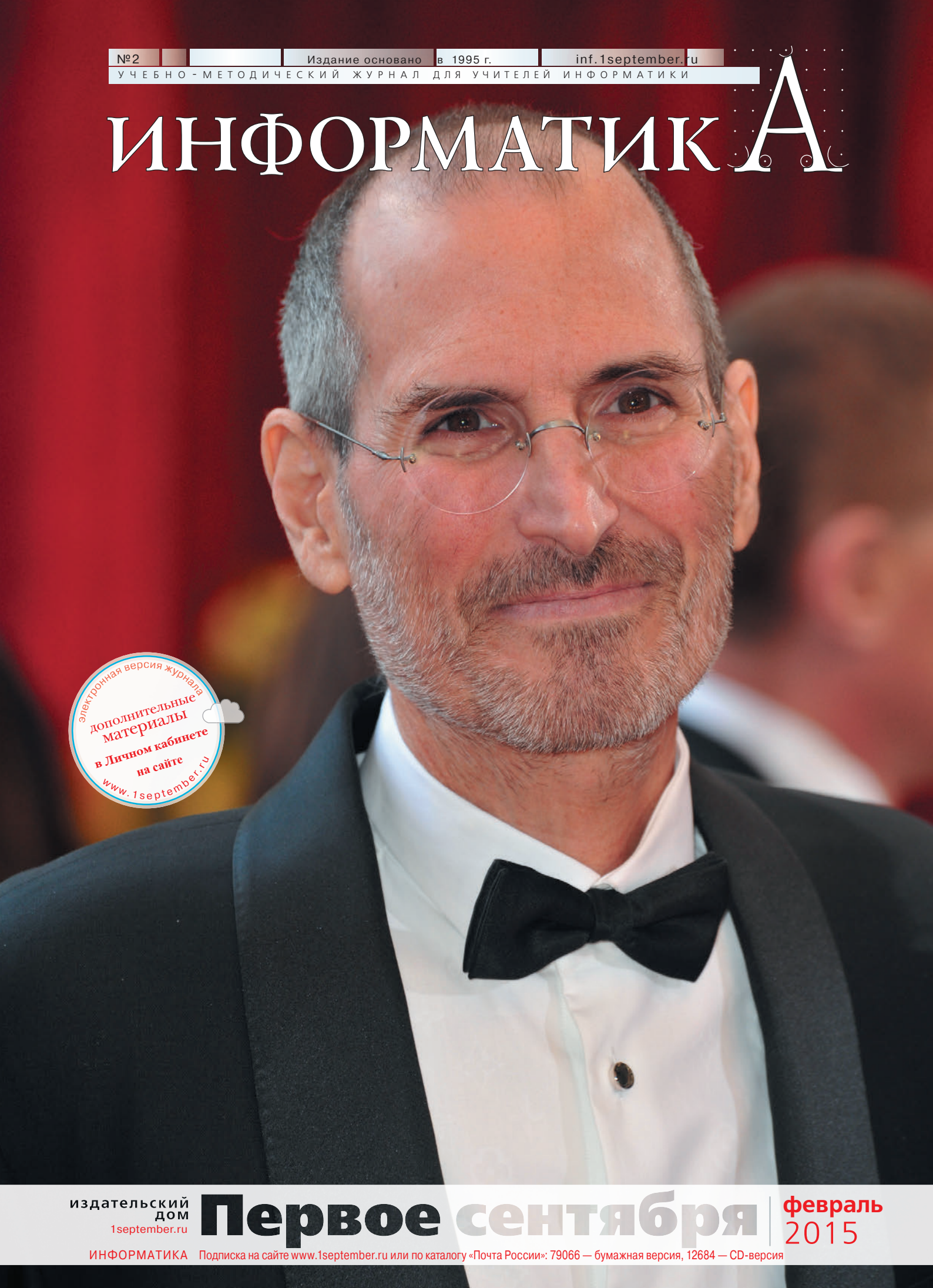
№2 Издание основано в 1995 г. inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА



электронная версия журнала
дополнительные материалы в Личном кабинете на сайте www.1september.ru

издательский дом 1september.ru
Первое сентября
февраль 2015
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

Нет таких слов, которые не были бы сказаны о Стиве Джобсе. Написаны книги, сняты фильмы. Рыночная стоимость Apple и сейчас, спустя четыре года после его смерти, превышает стоимость всего российского фондового рынка (это не камень в огород российского рынка, это лишь демонстрация сравнительного масштаба). Безумно жаль, что он ушел так рано. Как интересно — что бы он еще придумал, чем бы удивил, порадовал, разочаровал, разозлил, довел до бешенства. В любом случае не оставил бы равнодушным. Безумно жаль, что этого не будет. А ему ведь было бы только 60...

Featureflash / Shutterstock.com

В НОМЕРЕ

3 ПАРА СЛОВ
- Немного о флибустьерах...

4 ИНФОРМАЦИЯ
- Всероссийская научно-практическая конференция “Пропедевтика формирования инженерной культуры учащихся в условиях модернизации российского образования”

6 СЕМИНАР
- Волшебные миры Мартина Гарднера

14 УЧЕБНИКИ
- Алгоритмизация и программирование

48 ЗАНИМАТЕЛЬНЫЕ МАТЕРИАЛЫ ДЛЯ ПЫТЛИВЫХ УЧЕНИКОВ И ИХ ТАЛАНТЛИВЫХ УЧИТЕЛЕЙ
- “В мир информатики” № 205

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Уважаемые подписчики бумажной версии журнала!

Дополнительные материалы к номеру и электронная версия журнала находятся в вашем Личном кабинете на сайте www.1september.ru.

Для доступа к материалам воспользуйтесь, пожалуйста, кодом доступа, вложенным в № 1/2015.

Срок действия кода: с 1 января по 30 июня 2015 года.

Для активации кода:

- зайдите на сайт www.1september.ru;
- откройте Личный кабинет (создайте, если у вас его еще нет);
- введите код доступа и выберите свое издание.

Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.

ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ

Презентации к статьям номера

ИНФОРМАТИКА

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

http://inf.1september.ru

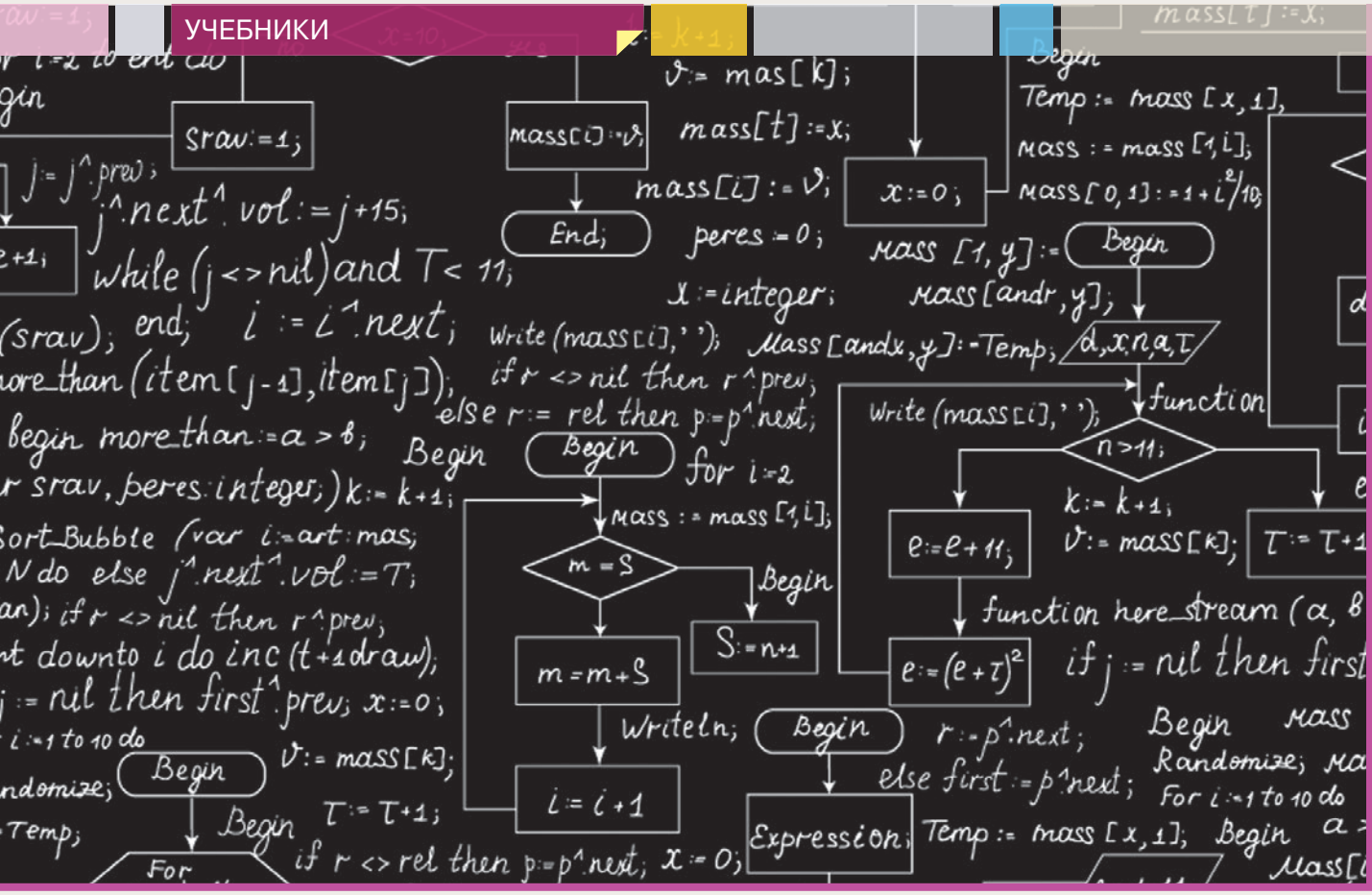
Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский
(редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 16 000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

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

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

УЧРЕДИТЕЛЬ:
ООО “ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»”
Зарегистрировано ПИ № ФС77-58447 от 25.06.2014 в Роскомнадзоре
Подписано в печать:
по графику 12.12.2014,
фактически 12.12.2014
Заказ №
Отпечатано в ОАО “Первая Образцовая типография”
Филиал “Чеховский Печатный Двор”
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76
АДРЕС ИЗДАТЕЛЯ:
ул. Киевская, д. 24,
Москва, 121165
Тел./факс: (499) 249-31-38
Отдел рекламы:
(499) 249-98-70
http://1september.ru
ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru



Алгоритмизация и программирование

К.Ю. Поляков,
д. т. н., Санкт-Петербург,
http://kpolyakov.spb.ru,
Е.А. Еремин,
к. ф.-м. н., г. Пермь

Алгоритмы и исполнители

Ключевые слова:
- алгоритм
- исполнитель
- система команд исполнителя
- дискретность
- понятность
- определенность
- конечность
- корректность

Что такое алгоритм?

Знаете ли вы, как сложить два многоразрядных числа? Наверняка знаете! В начальной школе вам рассказали, какие действия и в каком порядке нужно выполнить для того, чтобы решить эту задачу для любых чисел. Такую последовательность действий в информатике называют *алгоритмом*.

Это слово происходит от имени средневекового арабского ученого Мухаммада аль-Хорезми, который в IX веке описал правила вычислений с десятичными числами. Работы аль-Хорезми были переведены на латинский язык и стали известны в Европе. Через некоторое время слово “алгоритм” (от имени автора, которое по-латыни писали как *Algorizmi* или *Algorizmus*) стало обозначать любую систему вычислений по определенным правилам.

Мухаммад аль-Хорезми (IX век н.э.)

С алгоритмами мы постоянно встречаемся в жизни: все пошаговые инструкции, позволяющие, например, внести сумму денег на счет мобильного телефона или заказать билет на самолет через Интернет, — это алгоритмы.

С любезного разрешения авторов продолжаем публикацию глав из будущего (надеемся!) учебника.

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

В этом определении встретилось новое слово — “исполнитель”, то есть тот, кто выполняет алгоритм. Давайте разберемся, кто это такой.

Кто такой исполнитель?

Сможет ли маленький ребенок сходить в магазин за хлебом? Сможет ли собака оплатить счет за квартиру? Конечно, нет. Поэтому можно сделать вывод: любой алгоритм составляется для какого-то определенного исполнителя.

Исполнитель — это человек, животное или машина, которые могут понимать и выполнять некоторые команды.

Слова “некоторые команды” говорят о том, что исполнитель понимает только определенный набор команд. Компьютер, мобильный телефон и даже современная стиральная машина — это тоже исполнители со своим набором команд.

Система команд исполнителя (СКИ) — это набор команд, который понимает и умеет выполнять исполнитель.

Например, СКИ простейшего исполнителя Робот, который умеет только передвигаться по ячейкам клетчатой доски, содержит всего четыре команды: вверх, вправо, вниз и влево:

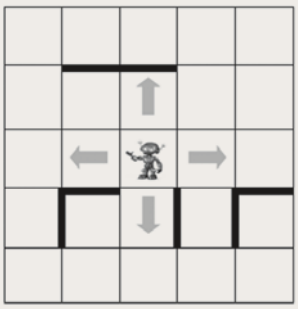


Рис. 1

Других команд этот Робот не понимает и выполнить не может.

На поле Робота могут встречаться стенки, которые обозначены жирными линиями. Робот не может ходить через стенки. Например, при попытке выполнить два раза подряд команду **вверх** Робот столкнется со стенкой и разрушится.

Любой исполнитель работает в некотором окружении (*среде*). Например, среда исполнителя Робот, с которым мы только что познакомились, — клетчатое поле со стенками.

Исполнители могут находиться в разных *состояниях*, и в зависимости от этого могут (или не могут) выполнять разные команды. Например, незаряженное ружье не выстрелит, а лежащий человек не сможет прыгнуть, не изменив свое состояние. Грузовик с пустым кузовом не сможет засыпать яму песком.

Свойства алгоритма

Дискретность — алгоритм состоит из отдельных команд (шагов), каждая из которых выполняется ограниченное время. Никакая команда не может быть выполнена раньше, чем закончится предыдущая.

Понятность — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен.

Определенность — при каждом выполнении алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат. Это значит, что исполнитель должен однозначно понимать команду и последовательность выполнения команд и выполнять их каждый раз одинаково.

Конечность (результативность) — любой алгоритм должен заканчиваться за конечное число шагов с некоторым результатом. Результатом может быть и сообщение о том, что задача не имеет решений.

Корректность — для любых допустимых исходных данных алгоритм должен приводить к правильному результату.

Массовость — алгоритм, как правило, предназначен для решения множества однотипных задач с различными исходными данными. Поэтому для составления алгоритма решение задачи нужно написать “в буквах”, вводя имена для исходных данных.

Эти свойства не равноправны. Дискретность, понятность и определенность — фундаментальные свойства алгоритма, то есть ими обладают все алгоритмы. Остальные свойства — это требования к “правильному” алгоритму.

Если работа алгоритма никогда не заканчивается, то говорят, что алгоритм *зациклился*. В этом случае результат его работы не определен.

Как правило, любую задачу можно решить с помощью различных алгоритмов. Например, найти сумму первых 100 натуральных чисел можно последовательным сложением:

$$1 + 2 + 3 + 4 + \ldots + 99 + 100 = 5050$$

Но когда эту задачу задали ученикам в одной немецкой школе, один из них сообразил, что если разбить все числа на пары (симметрично с противоположных концов), то все суммы

$$1 + 100, 2 + 99, \ldots 50 + 51$$

равны 101, поэтому общий результат равен $50 \times 101 = 5050$. Этим сообразительным мальчиком (согласно легенде) был великий математик Карл Фридрих Гаусс.

Какой же алгоритм выбрать? Конечно же тот, который лучше. Но что значит лучше? Обычно алгоритмы сравнивают по времени выполнения. При этом лучшим считается тот алгоритм, который быстрее приводит к правильному результату (требует меньше действий исполнителя). Кроме того, есть и другие критерии, например, количество памяти, которое требуется для работы алгоритма. Поэтому

выбор алгоритма зависит от особенностей задачи, которую вы решаете.

Контрольные вопросы

1. Откуда произошло слово “алгоритм”?
2. Что такое алгоритм?
3. Сформулируйте алгоритмы
а) сложения двух однозначных чисел;
б) вычитания однозначного числа из двузначного;
в) умножения двух двузначных чисел;
г) вычисления остатка от деления двух целых чисел;
д) вычисления среднего арифметического двух чисел.
4. Сформулируйте алгоритмы
а) заварки чая (как это делаете вы);
б) перехода через улицу по пешеходному переходу со светофором;
в) покупки бананов в магазине;
г) заправки автомобиля топливом;
д) оплаты мобильной связи через терминал.
5. Кого (что) называют исполнителем алгоритма?
6. Верно ли, что любой алгоритм составляют для какого-то определенного исполнителя? Докажите свою точку зрения.
7. Как вы думаете, можно ли считать алгоритмом рецепт приготовления блюда? Обоснуйте свою точку зрения.
8. Что такое “система команд исполнителя”?
9. Какие свойства алгоритма вы знаете? Какие из них обязательны?
10. Приведите примеры алгоритмов, работающих бесконечно.
11. Приведите примеры некорректных алгоритмов.
12. Что значит фраза “алгоритм зациклился”?
13. Как сравнивают различные алгоритмы решения одной и той же задачи?

Тема для сообщения:
“Аль-Хорезми и его вклад в науку”

Формальные исполнители

Ключевые слова:
- неформальный исполнитель
- формальный исполнитель
- непосредственное управление
- программное управление
- автоматизация
- Робот
- Черепаха
- Шифровальщик

Какие бывают исполнители?

Человек как исполнитель отличается от компьютера в первую очередь тем, что он может поразному выполнить одну и ту же команду или даже вообще отказаться ее выполнять. Например, пункт кулинарного рецепта “добавить соль по вкусу” каждый повар выполнит по-разному. Такой исполнитель называется *неформальным*, он может обдумывать команды и вносить в ход их выполнения свои поправки.

Во многих случаях такое поведение исполнителя недопустимо. Например, компьютер по команде сложения должен всегда выполнять именно сложение, а не вычитание или умножение.

> **Формальный исполнитель** — это исполнитель, который одну и ту же команду всегда выполняет одинаково.

Все машины, в том числе компьютеры и роботы, — это формальные исполнители. Иногда человеку тоже приходится быть формальным исполнителем, строго выполняя инструкции, например, во время чрезвычайных ситуаций или на военной службе.

Исполнители могут работать в двух режимах — в режиме управления с пульта (его называют также режимом *непосредственного управления*) и в программном режиме.

Режим *управления с пульта* означает, что в течение всего времени выполнения алгоритма кто-то управляет исполнителем, отдавая одну команду за другой. Исполнитель тут же выполняет каждую введенную команду. Так человек обычно управляет телевизором.

Для нас более интересно *программное управление*, когда в память исполнителя заранее записывается (на специальном языке) весь алгоритм, который ему нужно выполнить. После запуска этого алгоритма исполнитель выполняет все действия в автоматическом режиме. Это позволяет использовать *роботов* для автоматизации многих работ, которые раньше выполнял человек.

В информатике рассматривают только формальных исполнителей, поэтому дальше для краткости мы будем использовать термин “исполнитель” в значении “формальный исполнитель”.

Теперь рассмотрим для примера несколько исполнителей.

Исполнитель Робот

С исполнителем Робот мы уже познакомились. Обозначим команды, которые он понимает, числами:

```
1. вверх
2. вправо
3. вниз
4. влево
```

Алгоритм для Робота можно записать как последовательность номеров команд. Например, запись 23321 означает, что сначала Робот должен выполнить команду 2 (вправо), затем дважды команду 3 (вниз), потом еще один раз команду 2 и в конце команду 1 (вверх). Выполняя этот алгоритм, Робот пройдет из начального положения в клетку, отмеченную буквой А (*рис.* 2), не столкнувшись со стенками.

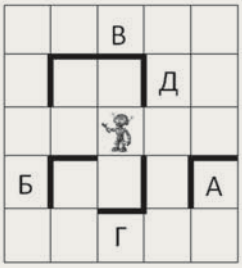


Рис. 2

Исполнитель Черепаха

Исполнитель Черепаха работает на плоскости, оставляя след при движении. Ее СКИ содержит такие команды:

- **вперед n** — передвинуться вперед на *n* шагов (здесь и дальше *n* — целое число);
- **назад n** — передвинуться назад на *n* шагов;
- **влево n** — повернуться на *n* градусов влево (против часовой стрелки);
- **вправо n** — повернуться на *n* градусов вправо (по часовой стрелке);
- **повтори n [...]** — повторить *n* раз все команды, записанные в квадратных скобках.

Черепаха поворачивается относительно своего *текущего направления* (то есть того направления, куда она смотрит в данный момент).

Например, алгоритм для Черепахи

```
повтори 4 [вперед 20 вправо 90]
```

заставляет ее нарисовать квадрат со стороной 20 шагов и вернуться в начальную точку, откуда она начала рисовать:

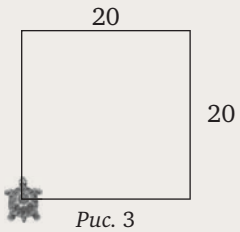


Рис. 3

Заметьте, что после выполнения алгоритма Черепаха вернется в начальное положение.

Исполнитель Удвоитель

Исполнитель Удвоитель работает с числами. Его СКИ состоит всего из двух команд:

```
1. прибавь 1
2. умножь на 2
```

Программа для Удвоителя — это последовательность номеров команд, записанная без пробелов. Например, программа 12211 означает, что сначала нужно выполнить команду 1, затем — дважды команду 2 и потом дважды команду 1.

Например, применим программу 12211 к начальному числу 2. На *рис.* 4 показаны все промежуточные результаты, которые исполнитель получает после каждого шага.

Над сплошными стрелками записаны номера выполняемых команд. В итоге получается число 14.

Программа:

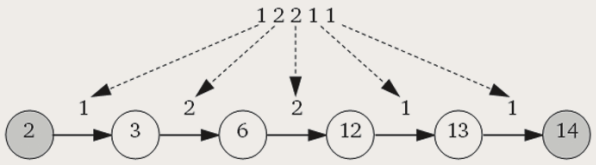


Рис. 4

Исполнитель Шифровальщик

Исполнитель Шифровальщик работает с цепочкой символов (символьной строкой), составленной из заглавных русских букв. Каждую букву строки он заменяет на следующую букву в алфавите[1]: А на Б, Б на В и т.д. Последнюю букву алфавита (Я) Шифровальщик заменяет на первую (А).

Например, после обработки строки ПРИВЕТ ВАСЯ получается РСКГЁУ ГБТА. Это шифровка, которую не сможет прочитать тот, кто не знает, как обработано сообщение.

Как построить исполнителя?

Давайте разберемся, что нужно для того, чтобы построить нового исполнителя.

В первую очередь необходимо определить его *среду* — обстановку, в которой он работает. Например, для Робота среда — это клетчатое поле, для Черепахи — плоскость.

Во-вторых, нужно задать набор команд исполнителя (СКИ) и определить, как именно он должен выполнять каждую из этих команд. Необходимо обдумать, в каких случаях исполнитель не сможет выполнить команду (например, Робот не сможет пройти сквозь стенку).

Контрольные вопросы

1. В каких случаях человек может отказаться выполнять команды? Когда он должен быть формальным исполнителем?
2. В каких случаях формальное выполнение алгоритма — это недостаток? Приведите примеры.
3. Чем отличаются режим управления с пульта и программное управление? Подумайте, в каких случаях лучше работает каждый из методов.

Задачи

1. Составьте две различные программы для Робота, которые переводят его из начальной клетки (*рис.* 2) в клетку Б. Робот не может проходить через стенки.
2. Выполните предыдущее задание для клеток В и Г.
3. *Сколько существует различных программ, которые переводят Робота из начальной клетки (*рис.* 2) в клетку Д?
4. *Может ли Робот перейти из начального положения в клетку Д за семь команд? Ответ обоснуйте.

[1] Шифр, который применяет Шифровальщик, называется шифром Цезаря.

5. Какую фигуру нарисует Черепаха, выполнив такие алгоритмы:

а) `повтори 4 [вперед 40 вправо 120]`
б) `повтори 4 [вперед 50 вправо 45]`
в) `повтори 4 [вперед 30 вправо 180]`
г) `повтори 6 [вперед 40 вправо 72]`
д) `повтори 10 [вперед 30 вправо 45]`
е) `повтори 10 [вперед 20 вправо 90]`

6. *Подумайте, как Черепаха может нарисовать окружность.

7. Какой результат получит Удвоитель, выполнив программу 211221 для начального числа

а) 2; г) 5; ж) 8;
б) 3; д) 6; з) 9;
в) 4; е) 7; и) 10?

8. Платон хочет с помощью исполнителя Удвоитель получить из числа 2 число 9. Сколько есть способов решения этой задачи?

9. *Дана программа для Удвоителя: 2112. Составьте программу из трех команд, которая равносильна заданной, то есть приводит к точно такому же результату для любого начального числа.

10. У исполнителя Калькулятор две команды:

```
1. прибавь 2
2. умножь на 2
```

Какое число получит этот исполнитель, если выполнит программу 122112 для начальных чисел:

а) 1; г) 4; ж) 7;
б) 2; д) 5; з) 8;
в) 3; е) 6; и) 9?

11. Платон хочет с помощью исполнителя Калькулятор из предыдущего задания получить из числа 2 число 14. Сколько есть способов решения этой задачи? Какие числа невозможно получить из числа 2 с помощью этого исполнителя? Почему?

12. У исполнителя Калькулятор две команды:

```
1. вычти 2
2. умножь на 3
```

Какое число получит этот исполнитель, если выполнит программу 211221 для начальных чисел:

а) 1; г) 4; ж) 7;
б) 2; д) 5; з) 8;
в) 3; е) 6; и) 9?

13. Исполнитель Шифровальщик работает с цепочкой символов (символьной строкой), составленной из заглавных русских букв. Каждую букву он заменяет на другую букву, которая стоит в алфавите через три буквы от исходной: А на Г, Б на Д и т.д. При этом букву Я Шифровальщик заменяет на В, букву Ю — на Б и букву Э — на А. Буквы Е и Ё считаются одинаковыми. Расшифруйте сообщения, закодированные с помощью этого алгоритма:

а) ПЛУЦ ПЛУ
б) УСФФЛВ ПСОСЗГВ
в) ЛФХСУЛВ ОБДЕЛ
г) ВЫГ БУНРЦО Е АООЛРЖ

14. Придумайте своего исполнителя: определите его среду, систему команд, способ выполнения каждой команды.

Темы сообщений:
1) “Исполнитель Кузнечик”
2) “Исполнитель Паркетчик”

Способы записи алгоритмов

Ключевые слова:
- словесная запись
- запись по шагам
- блок-схемы алгоритмов
- программа
- язык программирования
- ассемблер
- языки высокого уровня
- логическое программирование

Словесная запись

Как можно записать алгоритм? В первую очередь на естественном языке (русском, английском и др.). Это удобно и привычно, но есть одна проблема: во всех естественных языках есть неоднозначность, поэтому исполнитель-человек может понять алгоритм не так, как задумывал его автор. Особенно трудно разбираться в длинных словесных алгоритмах, занимающих больше десятка строк.

Пример 1. *Автомат получает на вход трехзначное десятичное число. По полученному числу строится новое десятичное число по следующим правилам. Сначала вычисляются сумма старшего и среднего разрядов, а также сумма среднего и младшего разрядов заданного числа. Затем полученные два числа записываются друг за другом в порядке невозрастания (без разделителей).*

Применим этот алгоритм к числу 277. Сначала находим сумму старшего и среднего разрядов: 2 + 7 = 9, и сумму среднего и младшего разрядов: 7 + 7 = 14. Выражение “в порядке невозрастания” означает, что второе число не должно быть больше первого. Так как 9 < 14, запись полученных чисел в порядке невозрастания выглядит так: 149.

Запись по шагам

Для того чтобы в алгоритме легче было разобраться, отдельные шаги принято записывать в виде нумерованного списка:

Пример 2.
Вход: два натуральных числа, a и b.
Шаг 1. Если $a < b$, перейти к шагу 4.
Шаг 2. Уменьшить a на величину b.
Шаг 3. Перейти к шагу 1.
Шаг 4. Стоп.
Результат: значение a.

Обратите внимание, что мы ясно определили, что служит исходными данными для работы этого алгоритма (см. первую строчку — *Вход*) и что будет считаться результатом его работы (строчка *Результат*). Воспринимать такую запись значительно проще, чем длинное словесное описание алгоритма, как в Примере 1.

Подумайте, что вычисляет этот алгоритм и что будет, если применить его к таким исходным данным: $a = 19$, $b = 5$.

Блок-схемы алгоритмов

Словесная запись удобна не во всех случаях. Например, если записать алгоритм на русском языке, то люди, не умеющие читать по-русски, не смогут им воспользоваться. Кроме того, возможно, вы заметили, что даже в простейшем алгоритме из четырех шагов непросто отслеживать переходы между шагами и разобраться, что же происходит с данными. Для более запутанных алгоритмов это становится *очень* сложно.

Поэтому придумали *графическую форму* записи алгоритмов — блок-схемы. Вот как выглядит блок-схема алгоритма из Примера 2 — см. *рис.* 5.

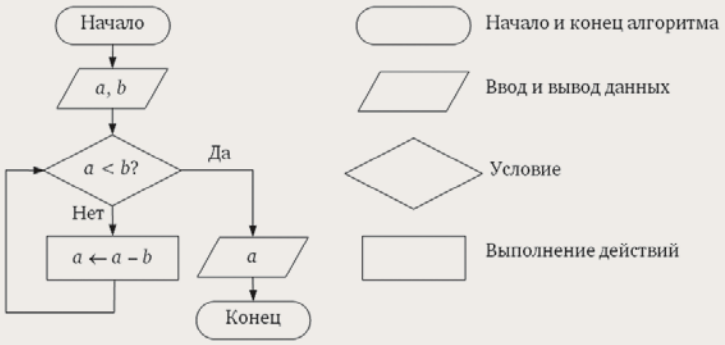


Рис. 5

В правой части рисунка показаны условные обозначения, которые используются в блок-схемах. Постарайтесь разобраться в этой схеме, используя словесную запись алгоритма, с которой мы работали раньше.

Основных блоков — четыре: начало и конец алгоритма, ввод и вывод данных, условие и процесс (действие). Внутри блока ▭ записывают данные, которые вводятся или выводятся, в блоке ◇ — условие, которое проверяется, а в блоке ▭ — выполняемое действие. Запись $a \leftarrow a - b$ означает "заменить a на $a - b$".

Стрелки между блоками показывают направление "движения" при выполнении алгоритма. Из блока ◇ ("условие") выходит две стрелки: если записанное внутри условие истинно (верно), нужно перейти по стрелке с надписью "да", а если неверно — то по стрелке с надписью "нет".

Языки программирования

Для того чтобы робот (или другой исполнитель) мог выполнять алгоритмы автоматически (без участия человека), он должен уметь работать в *программном режиме*. Это означает, что алгоритм нужно записать в память исполнителя на специальном языке, который он "понимает". Тогда алгоритм становится *программой*.

> **Программа** — это алгоритм, записанный на языке конкретного исполнителя.

Теперь остается только "запустить" программу, а дальше она будет выполняться автоматически. Для этого исполнитель должен уметь самостоятельно расшифровывать команды и определять, какая команда должна быть выполнена следующей. Это хорошо умеют делать компьютеры — универсальные устройства для обработки информации.

Программы для компьютеров составляются на специальных языках, которые называются *языками программирования*. Этим занимаются люди, профессия которых — *программист*.

Мы уже говорили о том, что все данные в компьютере хранятся в виде двоичных кодов. Программы также хранятся в памяти компьютера как двоичные коды. Но это не значит, что все программы обязательно составляются "на языке нулей и единиц".

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

Ближе всего к родному языку процессора так называемые *языки низкого уровня*[1], или *языки ассемблера*. В них каждая команда может быть напрямую переведена в соответствующую команду процессора. Например, вот так выглядит программа на языке ассемблера для самого обычного домашнего компьютера, которая складывает числа 5 и 6:

MOV AX, 5	1011 1000 0000 0101 000 000
MOV BX, 6	1011 1011 0000 0110 000 000
ADD AX, BX	0000 0011 1100 0011

Слева от вертикальной линии вы видите команды языка ассемблера[3], а справа — их перевод на "машинный язык", то есть на язык нулей и единиц. Программу, которая переводит символьную запись команд в цепочки битов, называют *ассемблером* (сборщиком).

На языке ассемблера обычно пишут программы, в которых нужно очень высокое быстродействие и небольшой размер — ядро операционной системы, драйверы устройств и т.п.

К сожалению, у каждого семейства процессоров свой язык ассемблера, поэтому программы, написанные для одного семейства, не подходят для другого. Чтобы решить эту проблему, разработали *языки высокого уровня*, которые приближены к естественному языку (чаще всего — к английскому).

Языки высокого уровня никак не связаны с компьютером, на котором будут выполняться.

[2] То есть наиболее близкие к компьютеру.

[3] Конечно, эти команды вам непонятны. Но сейчас важна только общая идея.

Тогда возникает вопрос — как же разные компьютеры будут понимать программы на этих языках? Решение было найдено: для каждого типа процессоров (и операционных систем) создается *транслятор* (переводчик) — программа, которая переводит текст программы, написанной на языке высокого уровня, в двоичные команды нужного процессора.

Первый язык программирования высокого уровня — Фортран — был разработан к 1957 году под руководством известного американского специалиста в области информатики Джона Бэкуса. Язык Фортран до сегодняшнего времени используется в научных расчетах.

В середине 1970-х годов Деннис Ритчи придумал язык Си, который остается одним из самых популярных языков программирования и сегодня. Достаточно сказать, что на нем написаны практически все операционные системы (в том числе *UNIX, Windows, Linux*). На основе языка Си разработано множество современных языков высокого уровня, в том числе C++, C# (читается как “Си шарп”), Java, Javascript и др. Именно эти языки чаще всего используются сегодня для создания программ.

Сейчас очень популярен язык Python, появившийся в 1991 году. Он широко используется в таких известных компаниях, как *Google* и *Яндекс*, применяется для программирования игр. Этот язык прост и содержит большую библиотеку алгоритмов для решения многих стандартных задач, с которыми встречаются программисты. Поэтому решение задачи на языке Python обычно занимает меньше времени, чем при использовании классических языков.

Страницы современных сайтов в Интернете тоже содержат программы. Для этой цели применяют, как правило, языки PHP и Javascript.

Языки Prolog и LISP были разработаны в 1970-х годах как языки для решения задач искусственного интеллекта. Программа на языке логического программирования Пролог, например, — это не алгоритм, а фактически только формулировка задачи. Нужно описать исходные данные, правила вывода (которые используются для решения) и определить цель (что нужно найти). Дальше Пролог-машина сама на основе полученных правил решает задачу, если это возможно. Сейчас эти языки используются редко, в основном — в среде научных работников.

Существуют языки высокого уровня, которые разработаны специально для обучения программированию. Долгое время в качестве первого языка программирования для начинающих использовался язык Бейсик (Basic), но сейчас он вытесняется другими языками.

Один из самых популярных учебных языков — Паскаль — придумал в 1970 году швейцарец Никлаус Вирт. Он назвал новый язык в честь французского физика Блеза Паскаля. Мы начнем изучать язык Паскаль в следующем году.

Со *школьным алгоритмическим языком*, который разработал в 1980-х годах академик А.П. Ершов, мы познакомимся через несколько уроков.

Всего существует несколько тысяч языков программирования, но только 20–30 из них широко используются на практике.

Контрольные вопросы

1. Сравните словесную запись алгоритмов в свободной форме, запись по шагам и блок-схему. Расскажите о преимуществах и недостатках каждой формы записи. Какая из них вам больше нравится? Почему?
2. Что вычисляет алгоритм в Примере 2?
3. Что означает запись $a \leftarrow a + 1$?
4. Какой блок на блок-схеме может иметь более одного входа и более одного выхода? Может ли он иметь один вход и два выхода? А два входа и один выход?
5. Как вы думаете, почему блоки ввода и вывода данных обозначаются на блок-схеме одинаково? Нравится ли вам такое решение?
6. Как вы думаете, почему блок-схемы не используются для записи сложных алгоритмов?
7. Поясните, чем отличаются термины “алгоритм” и “программа”.
8. Чем занимаются программисты? Как вы думаете, какими качествами должен обладать программист? Могла бы вам понравиться эта специальность?
9. Зачем нужны языки ассемблера? В чем их недостатки?
10. Как вы думаете, на каких языках сейчас пишут программы чаще всего: на языках ассемблера или на языках высокого уровня?
11. Как компьютер понимает программу, написанную на языке высокого уровня?
12. Чем отличаются языки логического программирования от других языков высокого уровня?
13. Как вы думаете, почему на практике широко применяются только несколько десятков языков программирования?
14. Какие достоинства и недостатки, на ваш взгляд, имеют языки программирования, разработанные специально для обучения?

Задачи

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

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

б) Полученные три числа записываются друг за другом в порядке убывания (без разделителей).

Выполните этот алгоритм для чисел

а) 835 и 196; в) 567 и 291;

б) 138 и 256; г) 239 и 871.

2. Какие из этих чисел не могут быть результатом работы алгоритма из задания 1:

а) 111;
б) 1965;
в) 13127;
г) 131511;
д) 1226;
е) 87?

3. Автомат получает на вход три двухзначных числа. По этим числам строится новое число по следующим правилам:

а) Вычисляются два числа — сумма старших разрядов заданных двухзначных чисел и сумма младших разрядов.

б) Полученные числа записываются друг за другом в порядке возрастания (без разделителей).

Выполните этот алгоритм для чисел:

а) 31, 29, 87;
б) 13, 82, 56;
в) 56, 72, 91;
г) 23, 98, 79.

4. Какие из этих чисел не могут быть результатом работы алгоритма из задания 3:

а) 12168;
б) 121321;
в) 9828;
г) 131511;
д) 1226;
е) 222?

5. Запишите алгоритм в пошаговой форме и в виде блок-схемы: “Дано натуральное число $a > 2$. Присвоить b значение 2 и проверить, делится ли a на b. Если делится, то сделать вывод, что число a — составное. Иначе проверить делимость a на b, где b последовательно принимает все целые значения от 3 до $a - 1$, каждый раз увеличиваясь на 1. Если a не делится ни на одно из этих чисел, сделать вывод, что число a — простое”.

6. Запишите алгоритм в пошаговой форме и в виде блок-схемы: “Дано натуральное число $N > 1$. Присвоить a значение 1, затем присвоить величине d значение 2^a. Если $d \geq N$, завершить выполнение алгоритма и считать результатом a. Иначе увеличить a на единицу, присвоить величине d значение 2^a и перейти к предыдущему шагу”.

Что делает этот алгоритм? Докажите, что он не зациклится при любом натуральном N.

7. Дан алгоритм, записанный в пошаговой форме:

Ввод: натуральное число a.
Шаг 1. Присвоить b значение 1.
Шаг 2. Присвоить b значение $a \cdot b$.
Шаг 3. Присвоить a значение $a - 1$.
Шаг 4. Если $a > 0$, то перейти к **Шагу 2**.
Шаг 5. Стоп.
Результат: значение b.

Что вычисляет этот алгоритм? Нарисуйте его блок-схему. Что получится, если выполнить его для числа 4? Числа 5?

Темы сообщений:

а) “Какие бывают языки программирования?”

б) “Языки программирования с русскими командами”

в) “Академик А.П. Ершов”

Ручное выполнение алгоритмов

Ключевые слова:

- алгоритм
- формальный исполнитель
- ручная прокрутка
- условие
- переменная
- присваивание

Зачем это нужно?

Программистам часто приходится разбираться, почему не работает какой-то алгоритм, свой собственный или даже написанный другим человеком. Для этого нужно выполнить алгоритм вручную для каких-то исходных данных и сравнить результаты каждого шага с тем, что должно было получиться.

Нужно помнить, что мы работаем с формальными исполнителями, которые не могут обдумывать команды, а просто их выполняют.

Выполнение алгоритма по словесной записи

Пример 1. Дан алгоритм обработки цепочки символов:

Сначала вычисляется длина исходной цепочки символов; если она нечетна, то дублируется средний символ цепочки символов, а если четна, то в начало цепочки добавляется буква Г. В полученной цепочке символов каждая буква заменяется буквой, следующей за ней в русском алфавите (А — на Б, Б — на В и т.д., а Я — на А). Получившаяся таким образом цепочка является результатом работы алгоритма. Что получится, если применить этот алгоритм к цепочке ПУСК?

Решение. Согласно алгоритму, найдем длину исходной цепочки ПУСК, она равна 4. Поскольку эта длина четна, в начало цепочки добавляем букву Г, получается ГПУСК.

Теперь каждая буква заменяется на следующую за ней в алфавите: Г ← Д, П ← Р, У ← Ф, С ← Т, К ← Л. Получаем ДРФТЛ.

Пример 2. Вернемся к алгоритму, который встретился нам в начале 0.

Вход: два натуральных числа, a и b.
Шаг 1. Если $a < b$, перейти к шагу 4.
Шаг 2. Уменьшить a на величину b.
Шаг 3. Перейти к шагу 1.
Шаг 4. Стоп.
Результат: значение a.

Вы уже наверняка догадались, что он служит для вычисления остатка от деления a на b с помощью вычитания.

Выполним вручную все шаги этого алгоритма при $a = 19$ и $b = 5$. Это называется *ручной прокруткой алгоритма*.

> **Ручная прокрутка** — это выполнение алгоритма человеком вручную.

Чтобы не держать в памяти все промежуточные результаты, составим таблицу, в которой будем за-

писывать все выполняемые шаги, а также изменения величин a и b:

	Действие		a	b
1			19	5
2	$a < b$?	нет		
3	$a \leftarrow a - b$		14	
4	$a < b$?	нет		
5	$a \leftarrow a - b$		9	
6	$a < b$?	нет		
7	$a \leftarrow a - b$		4	
8	$a < b$?	да		
9	Стоп			

В этой таблице четыре столбца: действие, которое выполняется (или проверяемое условие); результат проверки условия (выполняется или не выполняется?); изменение значений величин a и b. Для удобства строки пронумерованы. В строке 1 записаны начальные значения a и b.

Выполнение алгоритма начинается с шага 1 — проверки условия $a < b$ (строка 2 в таблице). Для начальных значений a и b условие не выполняется, поэтому перехода к шагу 4 нет. Далее выполняется шаг 2: значение a уменьшается на величину b, получается 14. Записываем это новое значение в третий столбец. Затем переходим к шагу 1, условие $a < b$ снова ложно, и т.д.

После того как уменьшение a выполнено трижды, значение a станет равно 4. При очередной проверке условие $a < b$ выполняется (истинно), происходит переход к шагу 4 (на команду Стоп) и алгоритм завершается.

Проверьте самостоятельно, что получится в результате работы этого алгоритма при $a = 15$ и $b = 19$.

Выполнение алгоритма по блок-схеме

Пример 3. Дана блок-схема алгоритма:

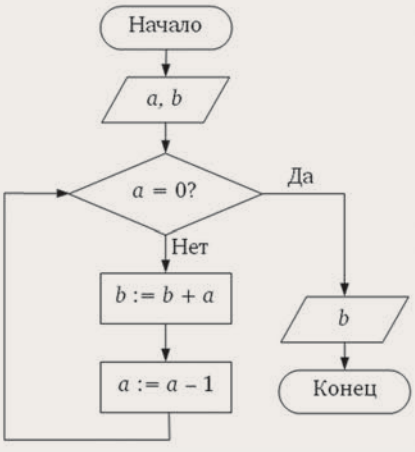


Рис. 6

Требуется определить результат работы этого алгоритма при входных значениях $a = 4$ и $b = 0$.

Наверное, вы заметили, что в двух блоках этой схемы используется новое обозначение ":=". Эти два символа во многих языках программирования обозначают присваивание. Операцию "заменить a на $a - 1$", которую мы до этого записывали как "$a \leftarrow a - 1$", можно записать еще и так: "$a := a - 1$". Именно второй способ используется, например, в школьном алгоритмическом языке и языке Паскаль, которые мы будем изучать в курсе информатики. Это связано с тем, что знака "←" нет на стандартной клавиатуре, и набирать ":=" значительно удобнее и быстрее.

Сначала нужно выполнить *анализ* алгоритма. Слово "анализ" обозначает "разделение на части", изучение внутреннего устройства. Нам необходимо определить:

1) какие исходные данные принимает алгоритм;
2) что будет результатом алгоритма;
3) как работает этот алгоритм.

В начале алгоритма вводятся исходные данные — значения a и b. После завершения работы алгоритма выводится результат — значение b.

На блок-схеме мы видим стрелку перехода к предыдущей команде (снизу вверх). Это означает, что две команды могут выполняться несколько раз: сначала b заменяется на сумму $b + a$, а затем a уменьшается на единицу. Когда значение a станет равно нулю, происходит выход из блока "условие" по ветке "да" на блок вывода результата, и работа алгоритма заканчивается.

Докажем, что алгоритм завершится (не будет работать бесконечно). Начальное значение a равно 4, и оно на каждом шаге уменьшается на 1. Поэтому через четыре шага a станет равно нулю и работа алгоритма завершится.

Выполним ручную прокрутку алгоритма, записывая все промежуточные данные в таблицу:

	Действие		a	b
1			**4**	**0**
2	a = 0?	нет		
3	b := b + a			**4**
4	a := a - 1		**3**	
5	a = 0?	нет		
6	b := b + a			**7**
7	a := a - 1		**2**	
8	a = 0?	нет		
9	b := b + a			**9**
10	a := a - 1		**1**	
11	a = 0?	нет		
12	b := b + a			**10**
13	a := a - 1		**0**	
14	a = 0?	**да**		

В первой строке записываем начальные значения a и b. Далее по блок-схеме прослеживаем ход выполнения алгоритма. Сначала проверяется условие (строка 2), оно ложно, поэтому во втором столбце пишем "нет".

Переходим по ветке "нет". Сначала выполняется присваивание $b := b + a$: определяем по таблице текущие (последние) значения величин a и b, скла-

дываем их и записываем в столбец изменения b (строка 3). Затем уменьшаем значение a на единицу (строка 4). После того как эти операции повторятся четыре раза, условие становится истинным ($a = 0$), и алгоритм завершает работу (строка 14).

Результат работы алгоритма — это значение величины b. Как мы видим из таблицы, последнее значение, присвоенное b, равно 10. Это и есть ответ.

Подумайте, что делает этот алгоритм. Для ответа на этот вопрос проверьте, что изменится, если начальное значение a будет равно 5, 6 и т.д.

При ручной прокрутке алгоритма мы видели, что величины a и b изменяются, им присваиваются новые значения. Такие величины в информатике называются *переменными*.

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

Операция, обозначаемая знаками ":=", — это присваивание нового значения переменной.

Контрольные вопросы

1. Объясните, зачем нужно уметь выполнять алгоритмы вручную.
2. Докажите, что алгоритмы в примерах 1 и 2 обязательно завершатся (не зациклятся).
3. Что такое "анализ алгоритма"? Где в жизни вам приходилось выполнять анализ ситуации?
4. Как по блок-схеме можно определить, что какие-то команды могут выполняться несколько раз?
5. Изменится ли результат работы алгоритма на 0, если поменять местами блоки с операциями $b := b + a$ и $a := a - 1$? Почему? Что произойдет, если их все-таки поменять местами?
6. Объясните, зачем нужны переменные при записи алгоритмов.

Задачи

1. Что получится, если применить алгоритм из примера 1 к словам
   а) УРА; в) КРОНА;
   б) НОРА; г) ЯРАНГА?
2. Что получится, если дважды применить алгоритм из примера 1 к словам
   а) РЕКА; в) РОСТОВ;
   б) МУРКА; г) МАЛЮТКА?
3. Дан следующий алгоритм: "*Сначала вычисляется длина исходной цепочки символов; если она четна, то в середину цепочки добавляется буква А, а если нечетна, то в начало цепочки добавляется буква Б. В полученной цепочке символов каждая буква заменяется буквой, следующей за ней в русском алфавите (А — на Б, Б — на В и т.д., а Я — на А). Получившаяся таким образом цепочка является результатом работы алгоритма*".

Что получится, если применить этот алгоритм к цепочкам
   а) СТАРТ; в) ХРЯМЗИК;
   б) МОЛОКО; г) КОМПЬЮТЕР?
4. *Цепочки символов были обработаны с помощью алгоритма из задания 3. В результате получились такие цепочки:
   а) НПТБЛГБ; в) ВРЕУЕСВФСД;
   б) ВАДПЕБ; г) ВРБСБЩЯУ.
   Определите исходные цепочки.
5. *Цепочки символов были дважды обработаны с помощью алгоритма из задания 3. В результате получились такие цепочки:
   а) ВОБВУР;
   б) ГПЖБТМВ;
   в) ВЩВДВЭЩВ;
   г) ГНРТБГХЪВ.
   Определите исходные цепочки.
6. Алгоритм из задачи 3 применен к слову четыре раза. В результате получилась цепочка из шести символов. Сколько символов было в исходной цепочке? Можете ли вы восстановить некоторые символы полученной цепочки?
7. Определите значение переменной b после выполнения этого алгоритма (здесь и далее для экономии места мы не будем рисовать блоки "начало" и "конец"):

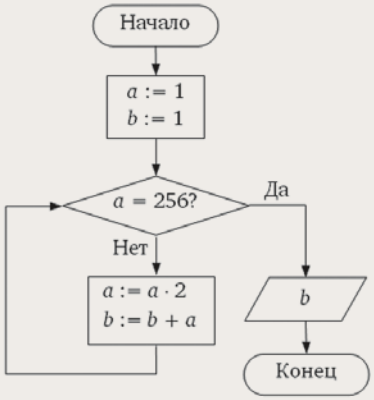


Рис. 7

8. Определите значение переменной b после выполнения следующего алгоритма:

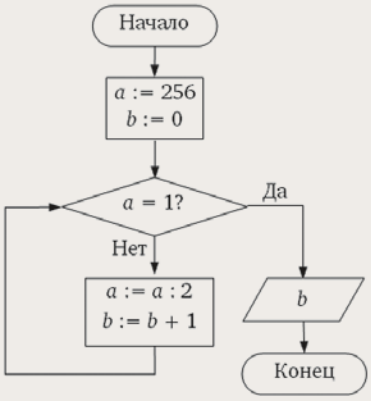


Рис. 8

9. Определите значение переменной b после выполнения следующего алгоритма:

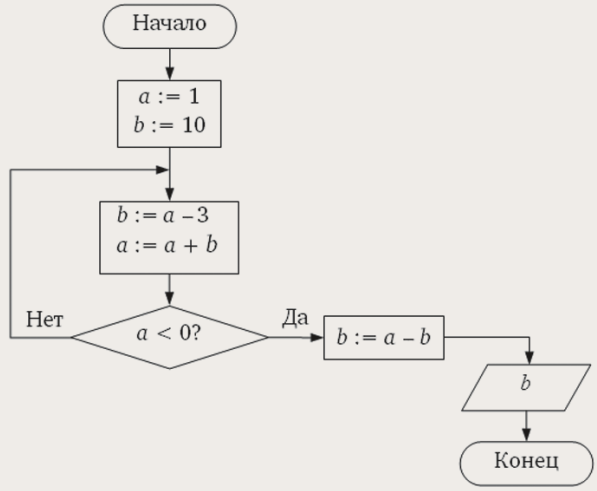


Рис. 9

10. *Определите значение переменной b после выполнения следующего алгоритма:

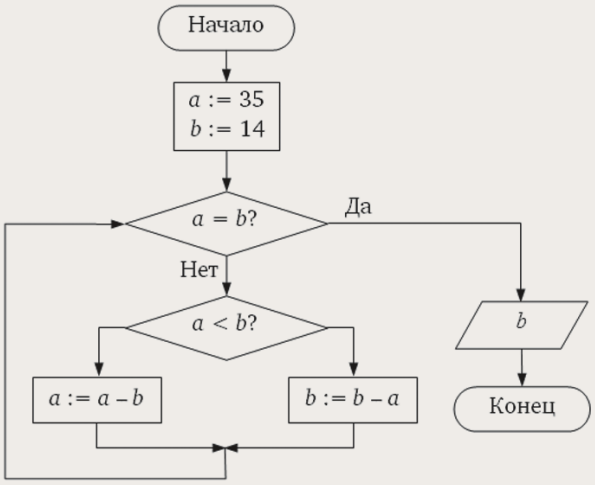


Рис. 10

11. *Определите значение переменной x после выполнения следующего алгоритма (знак “≥” обозначает “больше или равно”):

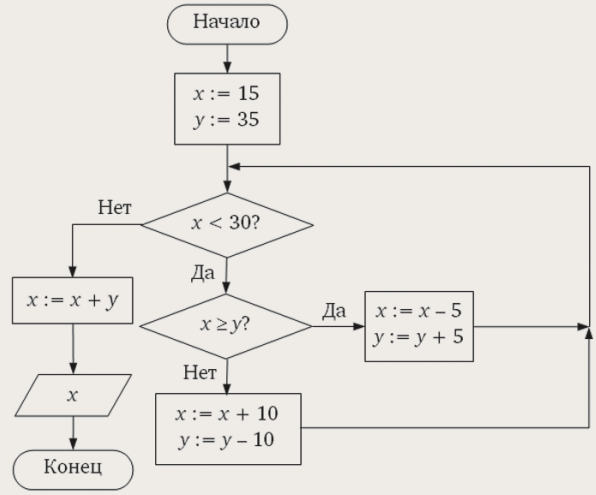


Рис. 11

Линейные алгоритмы

Ключевые слова:
- линейный алгоритм
- дерево вариантов

В этом и следующих параграфах мы познакомимся с различными типами алгоритмов и научимся их составлять.

Что такое линейный алгоритм?

В простейших алгоритмах все действия выполняются последовательно, одно за другим, при любых исходных данных. Пример такого алгоритма — вычисления по готовым формулам.

Задача 1. Сколько километров прошел автомобиль за $t = 2$ часа, если его скорость $v = 60$ км/ч?

Как мы уже говорили, одну такую задачу несложно решить на калькуляторе, а для быстрого решения серии однотипных задач лучше использовать компьютерную программу, например, электронную таблицу.

Алгоритм решения можно записать в словесной форме:

Вход: v, t.

Шаг 1. Присвоить S значение $v \cdot t$.

Результат: S.

Заметьте, что здесь нет ни одной команды перехода, которая направляет исполнителя на другой шаг. Поэтому порядок действий всегда одинаковый, какие бы исходные данные мы ни вводили.

Этот же алгоритм можно записать в виде блок-схемы:

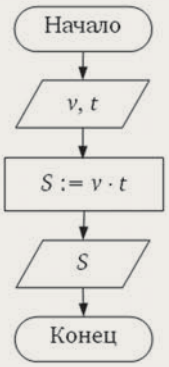


Рис. 12

Видим, что все блоки расположены “в одну линию”, которая идет между начальной и конечной точками алгоритма. Поэтому такой алгоритм называется *линейным*, он имеет единственный вариант исполнения.

> В **линейном** алгоритме команды выполняются в том же порядке, в котором они записаны.

Программы для Удвоителя

Задача 2. Составьте самую короткую программу для Удвоителя, которая преобразует число 6 в 28.

Напомним, что в СКИ Удвоителя всего две команды:

1. прибавь 1
2. умножь на 2

Решение задачи — это линейный алгоритм (последовательность команд).

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

Решение. Сначала проверим все программы, состоящие из одной команды. Таких программ всего две, в результате мы можем получить либо число 7 (если выполнена команда 1), либо число 12 (по команде 2):

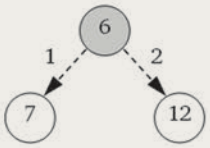


Рис. 13

Числа около стрелок показывают номер выполняемой команды. Таким образом, за один шаг можно получить только числа 7 и 12, никаких других чисел (в том числе заданного числа 28) получить нельзя.

Теперь рассматриваем все варианты программ из двух шагов. Из числа 7 на втором шаге можно получить 8 или 14, а из числа 12 — только 13 и 24.

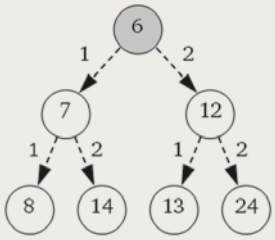
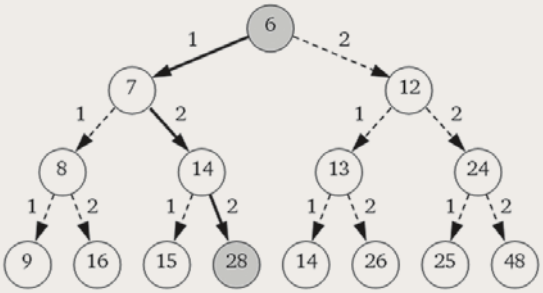


Рис. 14

Следовательно, за два шага получить число 28 тоже не удается.

Проверяем трехшаговые программы:



дерево

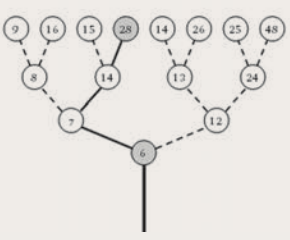


Рис. 15

На третьем шаге одно из полученных чисел — 28. Это значит, что самая короткая программа состоит из трех шагов. Чтобы построить саму программу, нужно пройти по стрелкам от начального числа к нужному результату (этот путь показан сплошной линией) и выписать номера всех встречающихся команд. В данном случае получаем 122.

Ответ: 122.

Схема, которую мы построили, называется *деревом возможных вариантов*. Действительно, мы рассмотрели все возможные программы, состоящие из одного, двух и трех шагов. Если полученный рисунок развернуть вверх ногами, он напоминает дерево.

Построение полного дерева вариантов при большой длине программы (например, 6 или 8 команд) — это очень утомительная работа. Во многих задачах бывает проще двигаться не от начального числа к результату, а наоборот.

Итак, возьмем конечное число 28 и подумаем, какой последней командой мы могли его получить. Так как 28 делится на 2, это могла быть как команда 1 (из 27 получаем 28), так и команда 2 (из 14 также получаем 28).

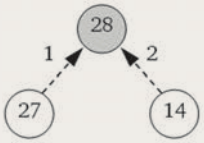


Рис. 16

Ни одно из этих чисел не совпадает с начальным (6), поэтому одной командой задачу не решить. Делаем второй шаг — проверяем, из каких чисел мы могли получить 27 и 14. Число 27 не делится на 2, поэтому оно могло быть получено только командой 1 (из 26), а число 14 можно получить из 13 или из 7:

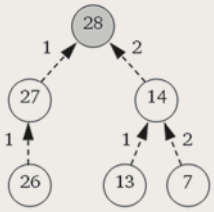


Рис. 17

До начального числа 6 снова добраться не удалось, поэтому решений из двух команд не существует. Делаем третий шаг: число 26 можно получить из 25 или из 13, число 13 — только из 12, и число 7 — только из 6 (а это и есть начальное число):

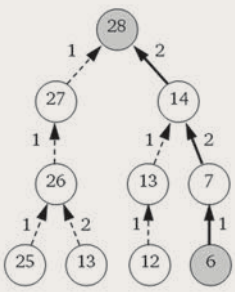


Рис. 18

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

Обратите внимание, что дерево при движении "в обратном направлении", от конечного числа к начальному, получилось меньше. Нам удалось найти решение быстрее, рассматривая меньше вариантов. Почему в некоторых случаях в дереве не было разделения на две ветки? Только потому, что не все числа делятся на 2, то есть не все числа могут быть получены командой 2.

Контрольные вопросы

1. Приведите примеры линейных алгоритмов, которые вы выполняете в жизни.
2. Какие алгоритмы, с которыми вы уже знакомы, нельзя считать линейными? Почему?
3. Как вы думаете, в чем недостаток линейных алгоритмов?
4. Может ли быть так, что задача для Удвоителя решается с помощью нескольких различных алгоритмов? Если да, приведите примеры.
5. Как можно доказать, что построенная программа для Удвоителя действительно самая короткая?
6. Какие числа можно получить из натурального числа N с помощью Удвоителя? Из нуля? Из отрицательного числа?
7. Как быстро построить самую короткую программу для получения некоторого числа N из нуля с помощью Удвоителя? Когда эта задача не имеет решений?
8. Исполнитель Калькулятор имеет команды

1. прибавь 1
2. раздели на 2

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

Задачи

1. Удвоителю нужно получить из числа 1 число 4. Сколькими способами это можно сделать? Выпишите все решения в виде программ для Удвоителя.
2. Удвоителю нужно получить из числа 3 число 12. Сколькими способами это можно сделать? Выпишите все решения в виде программ для Удвоителя.
3. Исполнитель Раздвоитель имеет команды

1. вычти 1
2. раздели на 2

Напишите самую короткую программу для Раздвоителя, которая преобразует

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

4. У исполнителя Калькулятор две команды:

1. вычти 3
2. умножь на 2

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 8 в число 14;
б) число 7 в число 19;
в) число 8 в число 17;
г) число 8 в число 40.

5. У исполнителя Калькулятор две команды:

1. прибавь 2
2. умножь на 3

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 5 в число 71;
б) число 5 в число 83;
в) число 4 в число 50;
г) число 4 в число 128.

6. У исполнителя Калькулятор две команды:

1. вычти 2
2. умножь на 3

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 6 в число 24;
б) число 7 в число 33;
в) число 8 в число 50;
г) число 10 в число 80.

7. * У исполнителя Калькулятор две команды:

1. вычти 4
2. умножь на 3

Напишите самую короткую программу для Калькулятора, которая преобразует

а) число 6 в число 22;
б) число 6 в число 118;
в) число 8 в число 22;
г) число 10 в число 34.

Программирование Робота

Ключевые слова:
- программа
- линейный алгоритм
- алгоритмический язык
- ключевые слова языка
- синтаксические ошибки
- отказы
- логические ошибки

Как оформить программу для Робота?

Вы уже знакомы с исполнителем Робот, который умеет ходить по клетчатому полю. Теперь мы научимся составлять для Робота настоящие программы (программировать исполнителя) в системе КуМир (*Комплект Учебных Миров*).

Оказывается, Робот может не только ходить по клетчатому полю, но и выполнять полезную работу — закрашивать клетку, в которой он стоит. Для этого используется команда

закрасить

Клетки, которые нужно закрасить по заданию, мы будем обозначать серой точкой в правом нижнем углу. Например, вот такая задача для Робота:

Рис. 19

решается следующим *линейным* алгоритмом:

```
вниз
закрасить
влево
закрасить
```

Для того чтобы Робот смог выполнить программу, нужно оформить ее так, чтобы она была понятна системе КуМир. Программа для решения задачи на *рис.* 19 выглядит так:

```
использовать Робот
алг Переход
нач
  вниз
  закрасить
  влево
  закрасить
кон
```

В первой строчке мы подключаем исполнителя. В КуМире несколько исполнителей (Робот, Черепаха, Чертежник, Рисователь), и в начале программы необходимо сообщить, для какого исполнителя написана эта программа. Любая программа для Робота должна начинаться строчкой

использовать Робот.

Программа на алгоритмическом[4] языке начинается словом **алг**, за которым записывают (необязательное) имя алгоритма. Мы назвали его *Переход*. Алгоритм начинается словом **нач** (сокращения от слова *начало*) и заканчивается словом **кон** (сокращение от слова *конец*). Между словами **нач** и **кон** записывают все команды, которые должен выполнить исполнитель. Видим, что в данном случае нет никакой возможности пропустить какую-то команду или выполнить команды не по порядку. Значит, это линейный алгоритм.

Слова **алг**, **нач** и **кон** — это ключевые слова алгоритмического языка. Они всегда имеют единственное значение, которое задали им разработчики языка. Использовать их в других ситуациях (придавать другой смысл) нельзя.

[4] Для краткости мы будем называть школьный алгоритмический язык просто “алгоритмический язык”.

Ключевые слова — это специальные слова языка программирования, имеющие единственное заранее определенное значение.

Для проверки нужно набрать программу в среде КуМир и запустить исполнителя, нажав клавишу F9.

Какие могут быть ошибки?

При написании программ неизбежны ошибки. Это связано с тем, что возможности человека, к сожалению, ограниченны, и даже хороший программист редко может написать сразу без ошибок программу длиннее, чем 40–50 строчек. Бояться ошибок не нужно, нужно научиться обнаруживать и исправлять их.

Прежде всего нужно понимать, что ошибки бывают разные:

1) **синтаксические** ошибки (ошибки типа “не понимаю”) — неверное написание команды. Например, если вместо слова **закрасить** вы напишете **закрась**, Робот не сможет понять такую команду, потому что ее нет в его СКИ (хотя для человека эти команды означают одно и то же);

2) **отказы** (ошибки типа “не могу”) — это ситуации, когда исполнитель понимает команду, но не может ее выполнить. Например, в этой ситуации Робот не сможет выполнить команду **вниз**, потому что под ним — стенка, Робот врежется в нее и разрушится:

Рис. 20

3) **логические** ошибки — это ошибки, которые исполнитель обнаружить не может: он понимает все команды, успешно выполняет их, но результат его работы не совпадает с тем, который нужен. Например, Робот не закрасил все клетки, которые нужно закрасить, или закрасил лишние.

Самое сложное — это обнаружить и исправить логические ошибки. Для этого часто приходится выполнять ручную прокрутку программы или выполнять ее в пошаговом режиме. Это означает, что мы выполняем команды не все сразу, а по одной, останавливая исполнителя после каждой выполненной команды. Так мы сможем определить, на каком шаге исполнитель сделал не то, что мы предполагали.

Пошаговый режим в системе КуМир включается нажатием клавиши F8. Если нажать клавишу F8 еще раз, исполнитель выполняет очередную команду (только одну!). После каждого шага цветным фоном выделяется команда, которая будет выполнена следующей.

Контрольные вопросы

1. Верно ли, что для каждой задачи существует единственный алгоритм решения? Ответ обоснуйте.

2. Как можно сравнить два различных алгоритма решения одной и той же задачи? Как выбрать лучший их них?

3. Два друга по-разному ищут ошибки в программах. Кирилл, написав программу, сразу запускает ее для того, чтобы транслятор обнаружил все синтаксические ошибки. Даниил же сначала внимательно изучает текст программы и пытается найти ошибки сам, а потом уже запускает ее на выполнение. Чем хорош каждый из методов?

4. Чем отличаются синтаксические ошибки и отказы?

5. Какие способы поиска логических ошибок в программах вы знаете?

6. К какому типу ошибок относится зацикливание алгоритма?

Задачи

1. Найдите другой способ решения задачи на *рис.* 19.

2. Сколькими способами Робот может перейти в клетку Б, если ему разрешено выполнить не более четырех команд?

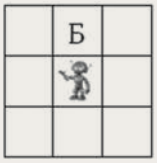


Рис. 21

Можно ли перейти в эту клетку ровно за четыре команды? За 2000 команд? за 2015 команд?

3. Напишите программы для Робота, которые решают следующие задачи (Робот должен закрасить все отмеченные клетки и прийти в клетку Б):

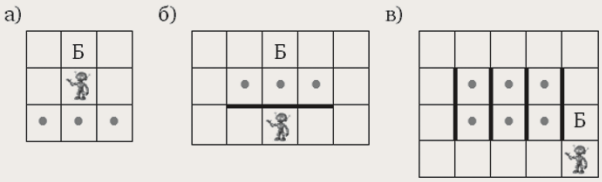


Рис. 22

Сравните ваши решения с решениями ваших одноклассников. У кого оказались самые короткие программы?

4. Тимофей написал программу для Робота, после выполнения которой исполнитель возвращается в начальное положение. Арсений похулиганил и стер одну команду. Помогите Тимофею восстановить ее:

```
влево
вверх
вверх
вправо
вниз
```

В каком месте программы могла быть стертая команда? От чего это зависит?

5. Тимофей написал программу для Робота, после выполнения которой исполнитель возвращается в начальное положение. Арсений похулиганил и стер одну команду. Но при запуске измененной программы Робот снова вернулся в начальное положение. Какую команду стер Арсений?

6. Тимофей написал программу для Робота, после выполнения которой исполнитель возвращается в начальное положение. Арсений похулиганил и переставил местами две команды в программе. Что может произойти при запуске этой программы?

7. Тимофей написал программу для Робота, после выполнения которой исполнитель возвращается в начальное положение. Арсений похулиганил и стер несколько команд. Помогите Тимофею восстановить ее:

```
влево
влево
вверх
вправо
вверх
```

Какое минимальное количество команд нужно добавить, чтобы восстановить программу? Какие это команды?

8. Тимофей написал программу, при выполнении которой Робот движется из клетки А в клетку Б:

```
влево
вверх
закрасить
вверх
вправо
закрасить
вниз
```

Напишите программу, которая переводит Робота в обратном направлении, из клетки Б в клетку А. Робот должен закрасить те же самые клетки.

9. Тимофей написал программу для Робота, при выполнении которой исполнитель закрашивает три клетки. Арсений похулиганил и поменял местами две команды в программе. Может ли новый алгоритм закрашивать

а) 0 клеток;

б) 2 клетки;

в) 4 клетки;

г) 10 клеток?

Тема для сообщения:

“Роботы вокруг нас”

Вспомогательные алгоритмы

Ключевые слова:

- вспомогательный алгоритм
- процедура
- вызов процедуры
- возврат из процедуры
- проектирование “снизу вверх”
- проектирование “сверху вниз” (метод последовательного уточнения)

В этом параграфе вы узнаете, как научить исполнителя выполнять новые команды.

Что такое вспомогательный алгоритм?

В этой задаче Роботу требуется закрасить три квадратных участка размером 2 × 2 клетки:

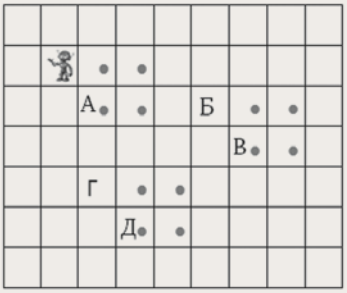


Рис. 23

Мы видим, что Роботу нужно будет трижды выполнить одинаковые команды. Наверное, вы подумали, что было бы хорошо, если бы у Робота уже была команда, которая сразу закрашивает квадрат размером 2 × 2? Тогда бы мы просто три раза использовали эту команду.

Хотя такой команды нет, мы можем ее ввести сами, расширяя СКИ исполнителя. Но при этом мы должны объяснить Роботу, что он должен делать, выполняя эту команду.

Итак, введем новую команду — **Квадрат**, выполняя которую Робот закрасит квадрат размером 2 × 2 клетки и остановится в его левом нижнем углу. Нарисуем условие этой вспомогательной задачи (Робот должен прийти в клетку А):

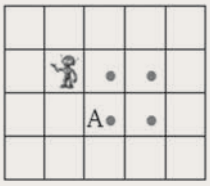


Рис. 24

Мы можем прямо сейчас написать ее решение:

```
алг Квадрат
нач
  вправо; закрасить
  вправо; закрасить
  вниз; закрасить
  влево; закрасить
кон
```

Здесь в каждой строке записаны две команды, их нужно разделять точкой с запятой.

Обратите внимание, что мы должны обязательно определить, где будет находиться Робот перед началом работы (слева от левого верхнего угла квадрата) и где он остановится после завершения работы (в правом нижнем углу квадрата).

> **Вспомогательный алгоритм** — это новая команда, которой мы дополняем СКИ исполнителя.

Как использовать вспомогательный алгоритм?

Теперь используем новую команду Квадрат для решения задачи на *рис.* 23. Видим, что Робот стоит в удачном исходном положении, так что можно сразу применить команду **Квадрат**. После ее выполнения Робот остановится в клетке А. Для того чтобы снова можно было использовать команду **Квадрат**, его нужно перевести в клетку Б:

```
Квадрат
вправо; вправо; вправо
```

Дальше снова применяем новую команду **Квадрат** и выводим Робота из клетки В, где он остановится, в клетку Г. После этого используем команду **Квадрат** в третий раз. Робот обрабатывает последний квадрат и останавливается в клетке Д:

```
Квадрат
вниз
влево; влево
влево; влево
Квадрат
```

Получается такая основная программа (приведем ее целиком):

```
алг Три квадрата
нач  Квадрат
  вправо; вправо; вправо
  Квадрат
  вниз
  влево; влево
  влево; влево
  Квадрат
кон
```

Не забудьте, что в первой строчке программы нужно подключить исполнителя командой

```
использовать Робот
```

Как оформить программу?

Теперь, для того чтобы все сработало, необходимо *ниже основного алгоритма* написать объяснение (расшифровку) команды **Квадрат**. Это *вспомогательный алгоритм*, который часто называют *процедурой*. Обратите внимание, что имя процедуры должно быть точно такое же, какое мы использовали в основной программе, причем заглавные и строчные буквы различаются, то есть **квадрат** и **Квадрат** — это разные имена.

Давайте разберемся, что будет происходить при запуске этой программы. Алгоритм, записанный в самом начале программы (у нас это алгоритм **Три квадрата**), называется основной программой. Исполнитель выполняет именно основную программу. В первой же строке он встречает неизвестную команду **Квадрат**, которая не входит в его СКИ. Это *вызов вспомогательного алгоритма*, или *вызов процедуры*.

> Для того чтобы вызвать вспомогательный алгоритм (процедуру), нужно записать его название в тексте другого алгоритма.

Встретив такой вызов, исполнитель ищет процедуру с названием **Квадрат** ниже основной программы. Если процедура **Квадрат** найдена, исполнитель выполняет все команды, записанные в теле этого вспо-

могательного алгоритма[5]. Затем происходит *возврат из процедуры* — передача управления на ту команду в основной программе, которая записана сразу после команды **Квадрат** (это команда **вправо**).

> После завершения работы вспомогательного алгоритма управление передается обратно, к следующей команде вызывающей программы.

Далее выполняются команды

```
вправо; вправо; вправо
```

(они входят в СКИ и не требуют расшифровки), а потом исполнитель снова встречает вызов процедуры **Квадрат**, уже второй по счету. Он действует так же, как в первом случае.

Обратите внимание, что если в основной программе нет вызова процедуры **Квадрат**, она не выполнится ни разу, хотя и будет записана в тексте программы.

> Вспомогательный алгоритм выполняется только тогда, когда он вызван из основной программы.

В только что рассмотренной задаче мы сначала написали вспомогательный алгоритм, а затем использовали его при составлении основной программы. Такой метод называется проектированием **снизу вверх** — решение основной задачи "собирается" из уже написанных ранее вспомогательных алгоритмов, как из кубиков.

Метод последовательного уточнения

Существует и другой подход к разработке программ, который называется программированием **сверху вниз**, или методом **последовательного уточнения**. Исходная задача разбивается на части (подзадачи) и сначала записывается основной алгоритм. Он использует вызовы еще не написанных вспомогательных алгоритмов, каждый из которых решает свою подзадачу. Затем пишем вспомогательные алгоритмы, если нужно, так же разбивая их на части.

Покажем этот подход на примере еще одной задачи для Робота:

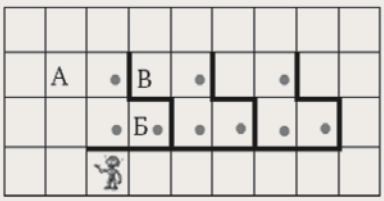


Рис. 25

Сразу видны три одинаковых блока из трех клеток, которые нужно закрасить Роботу. Робот расположен не очень удачно, так что в самом начале его нужно перевести в начальное положение рядом с первым блоком клеток, например, в клетку А. Назовем новую команду, которая это сделает, **Подход**.

Переход между первым и вторым блоками и между вторым и третьим выполняется одинаково. Мы введем еще две команды: **Блок** (обработка одного блока) и **Переход** (переход к следующему блоку). Теперь можно написать основную программу:

```
алг Блоки
нач
  Подход
  Блок
  Переход
  Блок
  Переход
  Блок
кон
```

Обратите внимание, что самих вспомогательных алгоритмов еще нет, но мы их уже используем, основная программа полностью готова.

Конечно, такая программа не будет работать, потому что исполнитель не знает, как выполнить команды **Подход**, **Блок** и **Переход**. Процедура **Подход** должна перевести Робота в клетку А:

```
алг Подход
нач
  влево; вверх; вверх
кон
```

Процедура **Блок** закрашивает блок из трех клеток; если Робот начал работу в клетке А, то он остановится в клетке Б:

```
алг Блок
нач
  вправо; закрасить
  вниз; закрасить
  вправо; закрасить
кон
```

А процедура **Переход** переводит Робота в исходное положение для обработки следующего блока (из клетки Б в клетку В):

```
алг Переход
  влево; вверх; вверх
  вправо; вниз
кон
```

Чтобы собрать программу, нужно сначала записать основной алгоритм, а за ним — все процедуры.

Во время отладки часто бывает нужно "войти" в процедуру и выполнить ее по шагам. Для этого в среде КуМир используется клавиша F7 (если вместо этого нажать клавишу F8, то процедура будет выполнена полностью).

Контрольные вопросы

1. Какие задачи легче решаются с использованием вспомогательных алгоритмов?
2. Можно ли использовать вспомогательные алгоритмы, если в программе нет повторяющихся действий? Зачем это может быть нужно?
3. В каких ситуациях вы не рекомендовали бы использовать вспомогательные алгоритмы?

[5] Как вы думаете, а что будет, если процедура **Квадрат** не будет найдена? Проверьте свою догадку в системе КуМир.

4. Что произойдет, если исполнитель не обнаружит расшифровки новой команды?

5. Где нужно записывать вспомогательные алгоритмы?

6. Что такое основная программа?

7. Вспомогательный алгоритм есть в тексте программы, но не срабатывает. В чем может быть причина?

8. Почему при составлении вспомогательного алгоритма нужно четко определить начальное и конечное положения исполнителя?

9. Можно ли записать в одной строке пять команд? Если да, как это сделать?

10. Как найти в программе вызов процедуры?

11. Как выполняется возврат из процедуры?

12. Куда передается управление, если процедура вызывается в самом конце основной программы (за ним нет других команд)?

13. Какой метод программирования вам больше нравится, “снизу вверх” или “сверху вниз”? Подумайте, какие достоинства и недостатки имеет каждый метод.

14. Как выполнить процедуру по шагам?

Задачи

1. Робот выполнил алгоритм с процедурой:

```
алг Площадка
нач
  к4
  вправо; вверх; вверх
  к4; вправо
  к4
кон
алг к4
нач
  закрасить
  вправо; закрасить
  вниз; закрасить
  влево; закрасить
кон
```

Ответьте на вопросы:

а) Как будет выглядеть площадка, закрашенная Роботом?

б) Сколько клеток закрасит Робот?

в) Сколько клеток будет закрашено дважды?

г) Сколько клеток будет закрашено трижды?

2. Составьте программы для Робота, решающие следующие задачи (Роботу нужно закрасить все отмеченные клетки и прийти в клетку Б):

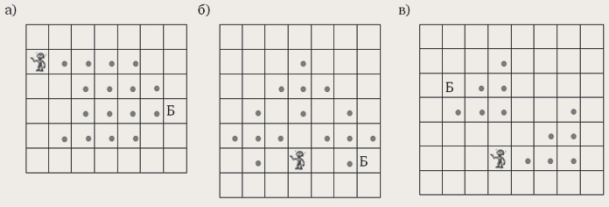


Рис. 26

Используйте вспомогательные алгоритмы.

3. Составьте программы для Робота, решающие следующие задачи (Роботу нужно закрасить все отмеченные клетки и прийти в клетку Б):

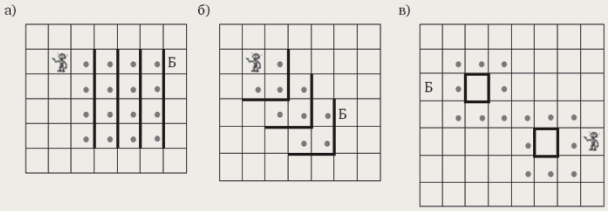


Рис. 27

Используйте вспомогательные алгоритмы:

Циклические алгоритмы

Ключевые слова:

- циклический алгоритм
- тело цикла
- вложенный цикл
- комментарий

Что такое циклический алгоритм?

Рассмотрим такую простую задачу для Робота:

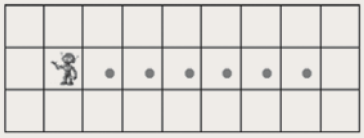


Рис. 28

Напомним, что серые точки в углах клеток означают, что эти клетки нужно закрасить.

Эту задачу можно решить с помощью линейного алгоритма:

```
вправо; закрасить
вправо; закрасить
вправо; закрасить
вправо; закрасить
вправо; закрасить
вправо; закрасить
```

Конечно, для запуска этой программы в среде КуМир нужно еще добавить команды **алг**, **нач** и **кон**, но для сокращения записи мы не будем повторять их каждый раз.

Теперь представьте себе, что Роботу нужно закрасить не шесть клеток, а 1006. Какой длины получится программа? Чтобы не повторять много раз одинаковые строчки в тексте программы, в языки программирования ввели специальные команды, которые говорят исполнителю, сколько раз нужно повторить те или иные действия. Такие команды называют *циклами*, а алгоритмы — *циклическими*.

> **Циклический алгоритм** — это алгоритм, в котором некоторая последовательность действий выполняется несколько раз.

В алгоритмическом языке существует цикл “*N* раз”, который записывают так:

```
нц 6 раз
  вправо
  закрасить
кц
```

Здесь **нц** обозначает “начало цикла”, а **кц** — конец цикла. Строчка **нц 6 раз** — это заголовок цикла, в котором указано число повторений.

Этот алгоритм можно записать в виде блок-схемы:

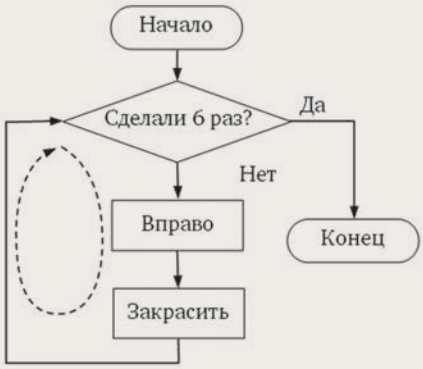


Рис. 29

Отличительная черта циклического алгоритма — замкнутый контур (петля), то есть стрелка, идущая к предыдущим командам (на этой блок-схеме — наверх). На *рис.* 29 эта петля обозначена штриховой линией.

Команды, которые выполняются в цикле (многократно), называются *телом цикла*. В нашем примере тело цикла состоит из двух команд: **вправо** и **закрасить**.

> **Тело цикла** — это команды, которые выполняются несколько раз.

Теперь немного усложним задачу: Роботу придется обходить стенки:

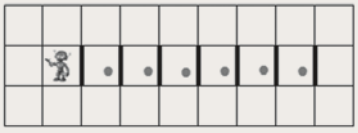


Рис. 30

Как изменится решение? Конечно, команда **закрасить** в конце тела цикла останется, но вместо команды **вправо** нужно использовать серию из трех команд:

```
вверх
вправо
вниз
```

Получается такое решение:

```
нц 6 раз
  вверх; вправо
  вниз;  закрасить
кц
```

Выбор начального положения

В только что решенных задачах Робот занимал очень удачную позицию: можно было сразу начинать цикл. Это не всегда так, чаще всего сначала приходится “подвести” исполнителя к клетке, с которой удобно начать действия в цикле.

В этой задаче Роботу нужно не только закрасить все отмеченные клетки, но и в конце работы прийти в клетку, отмеченную буквой Б (База).

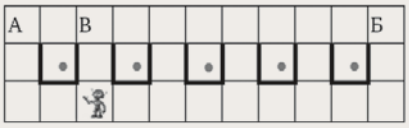


Рис. 31

Поскольку нужно закрасить пять клеток, расположенных в одну линию (хотя и через стенки!), хочется использовать цикл **нц 5 раз**. Но команды в теле цикла будут зависеть от начального положения Робота.

Конечно, можно начать прямо с той клетки, где стоит Робот. Но наиболее выгодная клетка для начала цикла отмечена буквой А. Тогда в теле цикла (на первом шаге) Роботу нужно будет перейти в клетку с буквой В, попутно закрасив клетку внизу, ограниченную стенками. Теперь Робот оказался в правильном исходном положении перед обработкой следующей клетки. Более того, после выполнения пяти шагов цикла он окажется в клетке с буквой Б, то есть вести его туда специально не придется.

Итак, программу можно составить, используя два вспомогательных алгоритма:

- прийти в клетку А (назовем его **Подход**);
- обработать клетку (он будет называться **Клетка**).

Получается такая основная программа:

```
Подход
нц 5 раз
  Клетка
кц
```

Вспомогательные алгоритмы (процедуры) **Подход** и **Клетка** нужно записать после основной программы:

```
алг Подход
нач
  влево; влево; вверх; вверх
кон
алг Клетка
нач
  вправо; вниз
  закрасить
  вверх; вправо
кон
```

При составлении этой программы мы использовали прием “программирование сверху вниз”: разбили задачу на подзадачи, составили основной алгоритм, а затем написали вспомогательные алгоритмы для решения подзадач.

Вложенные циклы

Рассмотрим задачу, в которой Роботу нужно закрасить поле размером 6 × 4 клетки:

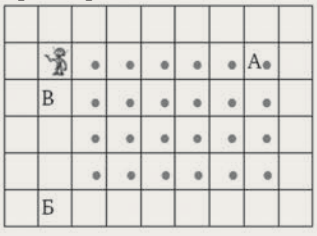


Рис. 32

Давайте сразу будем использовать “программирование сверху вниз” и напишем основную программу. Роботу нужно решать две подзадачи:

• закрасить ряд клеток (назовем эту процедуру **Ряд**);

• перейти к началу следующего ряда, например, из клетки А в клетку В (процедура **Переход**).

В основной программе можно использовать такой цикл:

```
нц 4 раза
  Ряд
  Переход
кц
```

Процедуры **Ряд** и **Переход** получаются очень простые:

```
алг Ряд
нач
  нц 6 раз
    вправо; закрасить
  кц
кон
алг Переход
нач
  вниз
  нц 6 раз влево кц
кон
```

Иногда решения простых подзадач не оформляют в виде коротких процедур, а включают прямо в основную программу. Это немного ускоряет работу программы и сокращает ее текст. Такой прием используют тогда, когда вызов процедуры встречается в программе только в одном месте.

Вот что получится в нашем случае, если встроить текст процедур **Ряд** и **Переход** в основную программу:

```
нц 4 раза
  нц 6 раз
    вправо; закрасить
  кц
вниз
  нц 6 раз влево кц
кц
```

Здесь фоном выделены два цикла, которые оказались внутри другого цикла. Такие циклы называются *вложенными*.

Вложенный цикл — это цикл, находящийся в теле другого цикла.

Но при таком объединении основная программа стала менее понятной, ведь мы ее значительно усложнили. Для того чтобы в программе было легче разобраться, в текст добавляют *комментарии* (пояснения), которые начинаются с символа “|”. Они не обрабатываются компьютером (он никак на них не реагирует) и служат только для того, чтобы человеку было легче понять программу. К нашей программе можно добавить два комментария (они выделены фоном) в тех местах, где начинается решение каждой из подзадач:

```
нц 4 раза
  | обработать ряд
  нц 6 раз
    вправо; закрасить
  кц
  | перейти в начало следующего ряда
  вниз
  нц 6 раз влево кц
кц
```

Комментарий — это пояснение к программе. Комментарии не обрабатываются исполнителем.

Комментарии в сложных программах очень важны, потому что очень часто основная работа программиста состоит в том, чтобы исправлять программы, написанные другими. Для этого нужно разобраться, как они работают, и это значительно легче сделать, если есть комментарии.

Напоследок давайте определим, где остановится Робот после выполнения программы. Вспомним, что в теле цикла решаются две подзадачи — обработка ряда и переход к началу следующего ряда. Это значит, что после выполнения всех шагов цикла Робот остановится в той клетке, откуда он мог бы начать обработку следующего, пятого ряда, если бы его нужно было закрашивать. Эта клетка отмечена буквой Б на *рис.* 32.

Контрольные вопросы

1. Подумайте, как Робот может определить, что он выполнил цикл нужное количество раз?

2. Как по блок-схеме определить, что алгоритм содержит цикл?

3. Когда можно включить текст вспомогательных алгоритмов в основную программу? Расскажите о достоинствах и недостатках такого решения.

4. Юный программист Семен говорит, что никогда не пишет комментариев в программах, потому что это лишняя работа, а он и так все помнит. Согласны ли вы с ним?

Задачи

1. Составьте программы для Робота, решающие следующие задачи (Роботу нужно закрасить все отмеченные клетки и прийти в клетку Б):

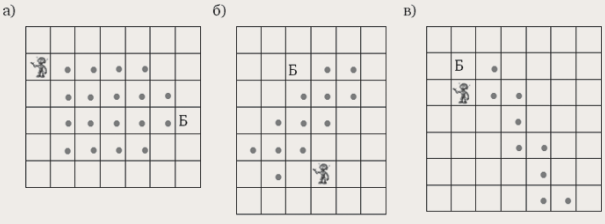


Рис. 33

Используйте циклы.

2. Составьте программы для Робота, решающие следующие задачи:

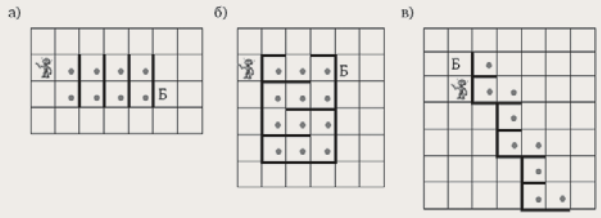


Рис. 34

Используйте циклы.

3. Составьте программы для Робота, решающие следующие задачи (Роботу нужно закрасить все отмеченные клетки и прийти в клетку Б):

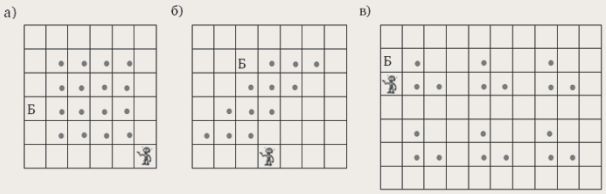


Рис. 35

Используйте вложенные циклы.

Циклы с условием

Ключевые слова:
- условие
- цикл с условием
- логические команды
- обратная связь
- зацикливание
- вложенные циклы

Что такое цикл с условием?

Вспомним алгоритм вычисления остатка от деления, о котором мы говорили ранее:

Вход: два натуральных числа, a и b.

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

Шаг 2. Уменьшить a на величину b.

Шаг 3. Перейти к шагу 1.

Шаг 4. Стоп.

Результат: значение a.

Как показано в разделе “Способы записи алгоритмов” (с. 18), исполнитель несколько раз выполняет шаги 1, 2 и 3 *с разными данными*. Вы уже знаете, что такой алгоритм называется циклическим.

В отличие от циклов, которые мы изучали в предыдущем параграфе, число шагов этого цикла заранее неизвестно, оно зависит от исходных данных. Поэтому мы не можем использовать цикл “N раз”.

Здесь цикл выполняется до тех пор, пока условие $a < b$ не станет истинным (тогда исполнитель перейдет к шагу 4 и работа алгоритма закончится). Можно сказать иначе: цикл выполняется, пока *обратное условие*, $a \ge b$, истинно. Это второй тип цикла, который называется *циклом с условием*.

> **Цикл с условием** — это цикл, который выполняется до тех пор, пока некоторое условие истинно. Количество шагов такого цикла определяется исходными данными.

Пусть Роботу нужно подойти к стене, которая находится слева от него, но неизвестно, на каком расстоянии:

Рис. 36

Если представить себя на месте Робота, можно решить задачу так: делать по одному шагу влево, проверяя, не дошли ли мы до стенки:

Шаг 1. Если слева стена, то перейти к шагу 4.

Шаг 2. Сделать шаг влево.

Шаг 3. Перейти к шагу 1.

Шаг 4. Стоп.

Можно нарисовать блок-схему, соответствующую этому алгоритму:

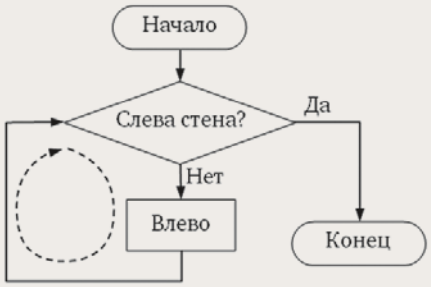


Рис. 37

Мы видим, что на этой блок-схеме есть замкнутый контур (петля), то есть стрелка, идущая к предыдущим командам. Это значит, что алгоритм *циклический*.

Как же исполнитель определит, что слева есть стена? Человек может увидеть ее или почувствовать, дотронувшись рукой. А у Робота тоже есть “органы чувств” — датчики. В систему команд Робота входят *логические команды*:

`сверху стена`	`сверху свободно`
`справа стена`	`справа свободно`
`снизу стена`	`снизу свободно`
`слева стена`	`слева свободно`

Это вопросы, на которые Робот (с помощью своих датчиков) отвечает “да” или “нет”.

> **Логическая команда** — это вопрос, на который исполнитель отвечает “да” или “нет”.

Логические команды также называют *командами обратной связи*, потому что информация передается в обратном направлении — не от управляющего устройства к исполнителю, а наоборот. С помощью этих команд Робот получает информацию об окружающей среде.

> **Обратная связь** — это данные, которые передаются от датчиков к управляющему устройству.

В нашей задаче цикл должен закончиться, когда слева есть стена. Другими словами, цикл работает, пока выполняется обратное условие — слева сво-

бодно. Поэтому решение на алгоритмическом языке запишется так:

```
алг До стены
нач
  нц пока слева свободно
    влево
  кц
кон
```

Здесь ключевые слова **нц** и **кц** обозначают, как и раньше, начало и конец цикла. После слова **пока** записано условие: цикл выполняется, пока это условие истинно (верно). Как только условие становится ложно, работа цикла заканчивается.

Обратите внимание, что использовать здесь цикл "*N* раз" нельзя, потому что мы не знаем точно, сколько шагов нужно сделать Роботу.

Если слева от Робота нет стены, он никогда не остановится, потому что условие **слева свободно** в заголовке цикла никогда не станет ложным. В этом случае говорят, что алгоритм *зациклился*. Чаще всего это означает, что он написан неверно.

Зацикливание — это ситуация, когда условие работы цикла никогда не становится ложным и цикл выполняется бесконечно.

В следующей задаче Роботу нужно закрасить ряд клеток, начиная от своего начального положения до стены справа:

Рис. 38

Здесь на каждом шаге цикла нужно выполнить две команды: **влево** и **закрасить**. Решение нашей задачи выглядит так:

```
нц пока слева свободно
  влево
  закрасить
кц
```

Заметим, что эта программа (как и предыдущая!) работает не для одной начальной обстановки, а для множества, потому что расстояние от Робота до стены справа может быть любым, Робот все равно закрасит все клетки.

Вложенные циклы

Циклы с условием можно вкладывать в другие циклы, так же, как и циклы "*N* раз". Пусть Роботу нужно закрасить четыре ряда клеток между двумя стенками, причем расстояние между этими стенками неизвестно.

Рис. 39

Выделим две подзадачи:

- закрашивание горизонтального ряда клеток (процедура **Ряд**);
- переход в начало следующего ряда (процедура **Переход**).

Тогда основная программа получается такая же, как и в решении задачи на *рис.* 32:

```
нц 4 раза
  Ряд
  Переход
кц
```

Однако здесь обе процедуры будут другие. Так как расстояние между стенками неизвестно, мы будем использовать циклы с условием.

Очевидно, что процедура "Ряд" будет содержать такой цикл:

```
нц пока справа свободно
  вправо
  закрасить
кц
```

Но при этом в каждом ряду не закрашивается клетка, находящаяся около левой стены (например, клетка, в которой стоит Робот в самом начале). Поэтому перед этим циклом нужно добавить еще одну команду — **закрасить**.

Процедура **Переход** включает движение влево до стенки (**пока слева свободно**) и шаг вниз, в начало следующего ряда:

```
нц пока слева свободно
  вправо
кц
вниз
```

Вместо того чтобы оформлять отдельные процедуры, мы можем включить все входящие в них команды в основную программу:

```
нц 4 раза
    | закрасить ряд клеток
  закрасить
  нц пока справа свободно
    вправо; закрасить
  кц
    | перейти в начало следующего ряда
  нц пока слева свободно
    влево
  кц
  вниз
кц
```

Мы получили вложенные циклы: внутри основного цикла "*N* раз" находятся два цикла с условием. Начало каждой из подзадач помечено комментарием.

Теперь еще усложним задачу — пусть рядов будет не четыре, а неизвестно сколько, причем эти ряды заканчиваются там, где заканчивается стена слева. Обратите внимание, что одна из этих задач — это задача с четырьмя рядами, которую мы рассматривали до этого.

Как изменится решение? Понятно, что нужно заменить цикл **нц 4 раза** на цикл с условием.

Определяя условие цикла, мы должны понять, когда его нужно остановить. В нашем случае цикл должен закончиться, когда закончится стенка слева. Это значит, что нужно продолжать цикл, пока выполняется *обратное условие*, то есть слева есть стена. Итак, просто заменяем одну строчку — заголовок основного (внешнего) цикла:

```
нц пока слева стена
  ...
кц
```

Вместо многоточия нужно добавить все команды, которые были внутри основного цикла в решении предыдущей задачи.

Контрольные вопросы

1. Объясните, в каких случаях нужно использовать цикл **N раз**, а в каких — цикл с условием.
2. Какая серьезная ошибка может встретиться при использовании цикла с условием?
3. Сработает ли алгоритм решения задачи на *рис.* 36, если Робот в самом начале уже стоит у стены?
4. Как по блок-схеме алгоритма определить, что он циклический?
5. Можно ли управлять исполнителем без обратной связи? В чем недостатки такого метода?
6. Цикл должен завершиться, когда кончится стена справа от Робота. Как записать заголовок такого цикла?
7. Можно ли включать циклы **N раз** в тело цикла с условием?

Задачи

1. Будет ли верно работать программа для решения задачи на *рис.* 39, если переход в начало следующего ряда выполнить так:

```
вниз
нц пока слева свободно
  влево
кц
```

Ответ обоснуйте.

2. Переведите Робота в клетку Б, считая, что все длины стенок неизвестны. Все отмеченные клетки нужно закрасить.

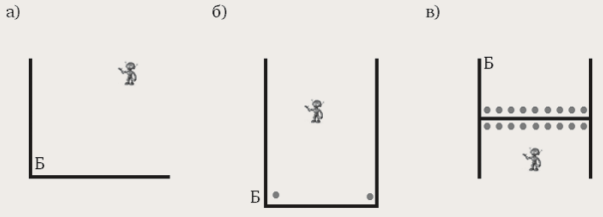


Рис. 40

3. Напишите программу, выполнив которую Робот закрасит все отмеченные клетки. Длины стенок считайте неизвестными.

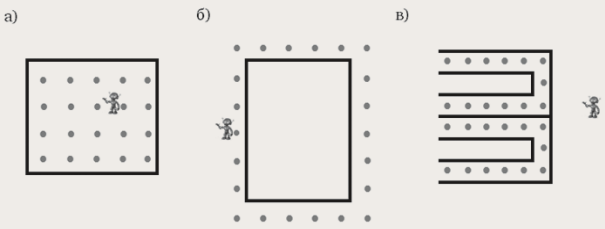


Рис. 41

Тема для сообщения:
“Сравнение двух видов циклов”

Переменные

Ключевые слова:
- переменная
- присваивание
- объявление переменной
- процедура
- параметр

Зачем нужны переменные?
Попробуем решить такую задачу:

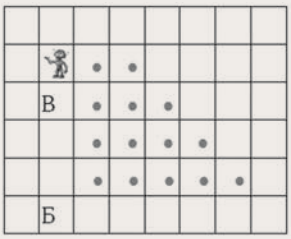


Рис. 42

Эта задача очень похожа на задачу на *рис.* 32. Но есть одно отличие: длины всех рядов тут разные: первый горизонтальный ряд состоит из двух клеток, а длина каждого следующего на одну клетку больше, чем предыдущего.

Обозначим длину очередного ряда буквой *N*. Тогда алгоритм закраски очередного ряда и возврата к началу следующего ряда выглядит так:

```
нц N раз
  вправо; закрасить
кц
вниз
нц N раз влево кц
```

Величина *N* должна изменяться во время работы программы, она называется *переменной*. С этим понятием мы уже знакомы. До начала основного цикла нужно присвоить этой переменной значение 2:

```
N := 2
```

Символ “:=” — это команда (*оператор*) присваивания, она присваивает переменной **N** значение, которое записано в правой части.

После обработки очередного ряда и возврата назад нужно увеличить значение *N* на 1, то есть заменить *N* на *N* + 1. Для этого тоже используется оператор присваивания:

```
N := N + 1
```

В результате получаем такую программу с вложенным циклом:

```
цел N
N := 2
нц 4 раза
    | закраска ряда длиной N
  нц N раз
    вправо; закрасить
  кц
    | переход к следующему ряду
  вниз
  нц N раз влево кц
  | увеличение длины ряда
  N := N + 1
кц
```

В ней есть одна неизвестная строка:

```
цел N
```

Это так называемое *объявление переменной*. Этой командой мы сказали, что в программе будет использована переменная с именем **N** и она может принимать только *целые* значения (ключевое слово **цел**).

Зачем нужно объявление переменной? При объявлении мы определяем

- какие данные можно хранить в этой переменной (целое число, дробное число, строка символов и др.);
- какие операции можно выполнять с переменной;
- сколько места в памяти компьютера нужно выделить для хранения значения.

Отметим, что в некоторых языках программирования, например в языке Python, объявлять переменные не нужно. С одной стороны, это облегчает составление программы, но с другой — может привести к труднообнаруживаемым ошибкам.

Процедуры с параметрами

Ранее мы уже познакомились с процедурами — вспомогательными алгоритмами, которые используются для того, чтобы не повторять в программе несколько раз одинаковые команды. Но до этого мы писали процедуры, каждая из которых всегда делала одно и то же. Часто бывает нужно несколько раз выполнить похожие, но чем-то отличающиеся действия. В этом случае тоже можно составить процедуру, но более сложного типа.

Рассмотрим такую задачу:

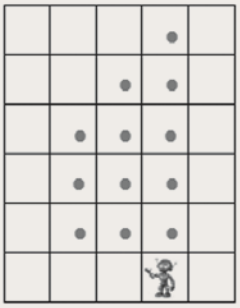


Рис. 43

Если бы все ряды были одинаковой длины (скажем, четыре клетки), мы могли бы написать процедуру (вспомогательный алгоритм) так:

```
алг Ряд
нач
  нц 4 раза
    вверх; закрасить
  кц
кон
```

Но длина ряда меняется, поэтому нужно сделать ее *переменной*, обозначив каким-то именем. Кроме того, надо как-то передать значение длины ряда из основной программы в процедуру (иначе как исполнитель узнает, какая длина ряда нужна именно сейчас?). При вызове мы хотели бы записать, например, так:

```
Ряд ( 4 )
```

надеясь, что исполнитель закрасит ряд из четырех клеток. Так действительно можно делать, но для этого в заголовке процедуры нужно указать, что она принимает *параметр* — значение, от которого зависит работа алгоритма.

В нашем случае параметр — это количество клеток, которые нужно закрасить. Если обозначить его именем *N*, получается такая процедура:

```
алг Ряд ( цел N )
нач
  нц N раз
    вверх; закрасить
  кц
кон
```

Обратите внимание, что в заголовке процедуры мы записали

```
Ряд ( цел N )
```

Это означает, что при вызове процедуры ей нужно передать (в скобках) целое число. Это значение будет записано во внутреннюю (или *локальную*) переменную процедуры с именем **N**.

> **Параметры** — это данные, которые передаются в процедуру. Каждый параметр имеет имя и тип.

Теперь можно полностью написать основную программу, которая решает задачу на *рис.* 43 с использованием процедуры **Ряд**:

```
алг Узоры
нач
  Ряд ( 5 )
  нц 5 раз вниз кц
  влево
  Ряд ( 4 )
  нц 4 раза вниз кц
  влево
  Ряд ( 3 )
кон
```

Попробуйте разобраться в ее работе самостоятельно. Не забудьте, что процедуру **Ряд** нужно записать ниже основного алгоритма.

Можно заметить, что после каждого выполнения процедуры **Ряд** исполнитель идет назад на столь-

ко шагов, на сколько он прошел вперед (на длину ряда), а затем делает шаг влево. Поэтому можно сократить основную программу, добавив в процедуру еще один цикл (“обратный ход”) и команду **влево**:

```
алг Ряд2 ( цел N )
нач
  нц N раз
    вверх; закрасить
  кц
  нц N раз вниз кц
  влево
кон
```

Теперь основная программа выглядит так:

```
алг Узоры2
нач
  Ряд2( 5 )
  Ряд2( 4 )
  Ряд2( 3 )
кон
```

Разница только в том, что в этом случае после выполнения алгоритма Робот остановится в другой клетке (подумайте, в какой?).

Числа, которые записаны в скобках при вызове процедуры **Ряд**, — это фактические значения параметров, которые также называют *аргументами*.

Контрольные вопросы

1. Можно ли решить задачу на *рис.* 42, не используя переменные?
2. Как, на ваш взгляд, компьютер может выполнить цикл **N раз** (как он запомнит, сколько раз уже выполнил тело цикла)?
3. Что означает запись **N := N + N**?
4. Зачем нужно объявлять переменные?
5. Сколько различных процедур без параметров мы могли бы написать для решения задачи на *рис.* 43?
6. Что значит “передать данные в процедуру”? Зачем это нужно?
7. Как по записи процедуры выяснить, принимает ли она параметры, и если да, то какие именно?
8. Объясните разницу между результатами работы алгоритмов **Узоры** и **Узоры2**.

Задачи

1. Определите, какую область закрашивает Робот, выполнив эти программы:

а)

```
цел N
N := 5
нц 3 раза
  нц N раз
    закрасить; вправо
  кц
  нц N - 1 раз влево  кц
  вниз
  N := N - 2
кц
```

б)

```
цел N
N := 1
нц 3 раза
  нц N раз
    закрасить; вправо
  кц
  нц N + 1 раз влево кц
  вниз
  N := N + 2
кц
```

Напишите программу, выполнив которую Робот закрашивает отмеченные клетки:

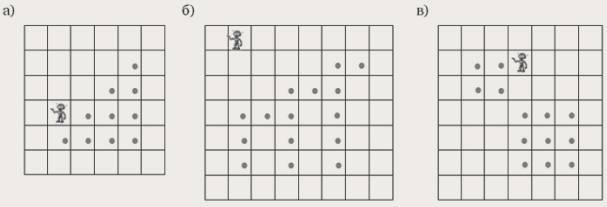


Рис. 44

Составьте два варианта каждой программы: один с вложенными циклами, второй — с процедурой, которая вызывается в цикле.

Разветвляющиеся алгоритмы

Ключевые слова:

- условие
- ветвление
- полная форма ветвления
- неполная форма ветвления
- разветвляющийся алгоритм
- вложенные ветвления

Что такое ветвление?

Очень часто в программе не обойтись линейными и циклическими алгоритмами, потому что исполнитель должен выполнять различные действия при разных исходных данных. Представим себе, что робот измеряет характеристики окружающей среды (например, радиацию) и в зависимости от результатов измерений либо выдает сигнал об опасности (если она больше допустимой), либо сообщает, что все нормально.

Пусть допустимый уровень радиации — 100 микрорентген в час[6]. Тогда алгоритм в словесной форме можно записать так:

Вход: *R*

Шаг 1. Если *R* > 100, перейти к шагу 4.

Шаг 2. Присвоить *M* значение “Все нормально!”.

Шаг 3. Перейти к шагу 5.

Шаг 4. Присвоить *M* значение “Тревога!”.

Шаг 5. Стоп.

Результат: *M*.

Здесь через *R* обозначен измеренный уровень радиации (в микрорентгенах в час), а *M* — это сообщение-результат.

[6] Это в 10 раз больше безопасного уровня, который составляет примерно 10–12 микрорентген в час.

Обратите внимание на две особенности этого алгоритма:

1) последовательность выполняемых команд зависит от исходных данных (результата измерения радиации);

2) в словесной форме запись этого простого алгоритма получилась довольно запутанной, и понять ее непросто.

Для представления алгоритма в виде блок-схемы нам понадобится блок “условие”:

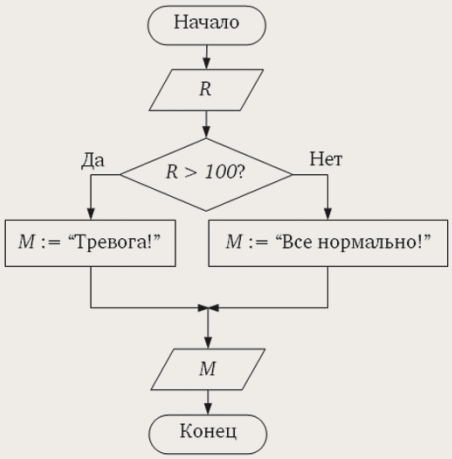


Рис. 45

Алгоритмы, в которых действия исполнителя зависят от исходных данных, называются *разветвляющимися*. На блок-схеме маршрут “расщепляется” на две ветки, появляются два пути выполнения алгоритма.

Разветвляющийся алгоритм — это алгоритм, в котором последовательность действий изменяется в зависимости от выполнения некоторых условий.

Разветвляющиеся алгоритмы позволяют исполнителю реагировать на изменение окружающей обстановки. Это значит, что он может решать более широкий круг задач.

Неполная форма ветвления

Обычно исполнитель должен выводить какие-то сообщения только в случае аварийной ситуации, а при нормальной работе никаких действий от него не требуется. Если бы ваш компьютер или смартфон каждую секунду выводил сообщение “Все нормально!”, это мешало бы работать с ним. Лишнюю информацию, которая только отвлекает человека от решения своих задач, часто называют “информационным шумом”. Желательно, чтобы такого шума было как можно меньше. Например, алгоритм проверки уровня радиации можно изменить так:

Вход: *R*

Шаг 1. Если $R \le 100$, перейти к шагу 4.

Шаг 2. Присвоить *M* значение “Тревога!”.

Шаг 3. Вывести *M*.

Шаг 4. Стоп.

Изменится и блок-схема:

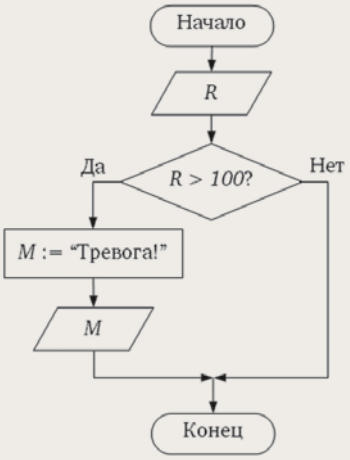


Рис. 46

Итак, если условие не выполняется, то исполнителю ничего делать не нужно, ветка “нет” — пустая! Такое ветвление называется *неполным*, а вариант, который мы использовали сначала, — *полной формой* ветвления.

Неполная форма ветвления отличается тем, что одна из веток на блок-схеме — пустая.

Управление Роботом

Ветвления — это очень мощное средство, позволяющее исполнителю ориентироваться в незнакомой (или не полностью известной) обстановке. С помощью одного и того же алгоритма с ветвлениями исполнитель может решать различные задачи.

Рассмотрим две задачи для исполнителя Робот, в которых требуется перевести его в соседнюю клетку, отмеченную буквой Б:

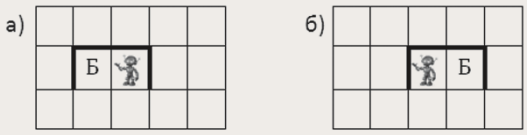


Рис. 47

Каждая из этих задач решается простым алгоритмом:

```
алг Случай а
нач
  влево
кон

алг Случай б
нач
  вправо
кон
```

Они отличаются единственной командой.

А теперь потребуем, чтобы обе задачи решались с помощью одного алгоритма для Робота. Для этого исполнителю нужно как-то решить, какую команду применить: **`влево`** или **`вправо`**.

В случае *а* справа от Робота будет стенка, поэтому туда идти нельзя, нужно идти влево. Если же справа стенки нет, нужно идти вправо (вариант *б*). Как вы уже знаете, в СКИ Робота есть логические команды, которые позволяют получать информацию от датчиков. В нашем случае нужна команда **справа стена**. Получается такая блок-схема алгоритма:

Рис. 48

А вот так запишется соответствующая программа для Робота на алгоритмическом языке:

```
алг В соседнюю клетку
нач
  если справа стена то
    влево
  иначе
    вправо
  все
кон
```

После слова **если** записано условие (вопрос). Если условие **справа стена** выполняется (то есть истинно), выполняются все команды между ключевыми словами **то** и **иначе**, а если не выполняется (ложно), то исполнитель выполняет все команды между словами **иначе** и **все**.

Еще раз подчеркнем, что только что построенный разветвляющийся алгоритм позволяет решать *обе* задачи на *рис.* 47.

Пусть теперь требуется перевести Робота в клетку Б, при этом сначала Робот может находиться как в клетке А, так и в клетке Б:

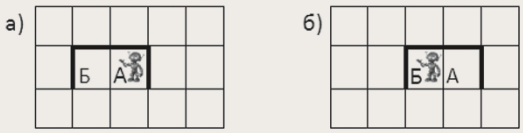


Рис. 49

В случае *а* Робот определит, что слева свободно, и передвинется влево. А в случае *б* исполнитель уже стоит в нужной клетке, поэтому ничего делать не надо. Наверное, вы догадались, что здесь нам нужна неполная форма ветвления:

```
алг К левой стенке
нач
  если слева свободно то
    влево
  все
кон
```

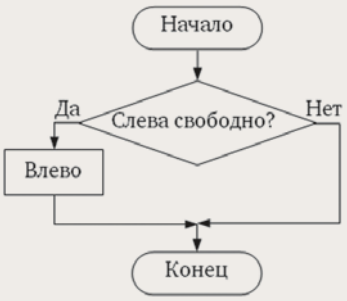


Рис. 50

Обратите внимание, что в записи программы на алгоритмическом языке нет слова **иначе**. Для ветвления в неполной форме второй блок команд (между словами **иначе** и **все**) не нужен, ведь он пустой! И ветка **нет** на блок-схеме тоже пустая.

Вложенные ветвления

Ветвления, как и циклы, могут быть вложенными. Составим алгоритм, который определяет знак числа. Результат его работы для некоторого числа x равен

- 1, если число x положительное;
- –1, если число x отрицательное;
- 0, если число x равно 0.

Вот блок-схема этого алгоритма:

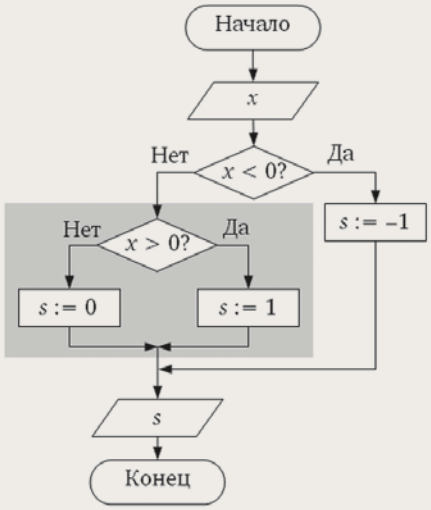


Рис. 51

Сначала проверяем условие $x < 0$. Если оно истинно (верно), то сразу определяем результат — он равен “–1”, записываем это значение в переменную s. Если же условие $x < 0$ ложно (неверно), то остаются еще два варианта, между которыми нужно выбрать. Поэтому внутри ветки “нет” должно быть еще одно ветвление (оно выделено на блок-схеме серым фоном).

Контрольные вопросы

1. Как человек использует разветвляющиеся алгоритмы в жизни? Приведите примеры.
2. Какие возможности появляются при использовании разветвляющихся алгоритмов? В каких задачах без них обойтись невозможно?

3. Как заменить полную форму ветвления на две неполных? Когда два ветвления в неполной форме можно заменить на одно ветвление в полной форме?

Задачи

1. Нарисуйте блок-схему алгоритма, который вычисляет модуль исходного числа.
2. Нарисуйте блок-схему алгоритма, который позволяет найти:
а) наибольшее из двух чисел;
б) наибольшее из четырех чисел;
в) наибольшее из трех чисел;
г) *наибольшее из пяти чисел.

Попробуйте найти решения этих задач, использующие только ветвления в неполной форме.

3. Нарисуйте блок-схему алгоритма, который позволяет расставить три числа в порядке возрастания (неубывания).
4. Нарисуйте блок-схему алгоритма, который позволяет определить, четное число x или нечетное. Нужно вывести ответ “да” или “нет”.
5. Нарисуйте блок-схему алгоритма, который позволяет определить, принадлежит ли значение x отрезку $[a,b]$. Нужно вывести ответ “да” или “нет”.
6. Нарисуйте блок-схему алгоритма, который позволяет определить, можно ли построить треугольник с заданными сторонами a, b и c. Нужно вывести ответ “да” или “нет”.
7. Нарисуйте блок-схему алгоритма, который выполняет деление одного числа на другое, обрабатывая ошибку “деление на ноль”.
8. Определите, что делают эти алгоритмы:

а)

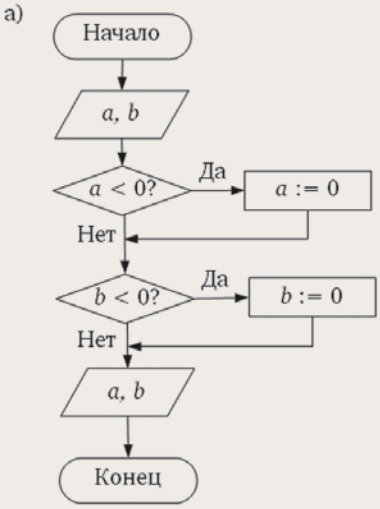


Рис. 52а

б)

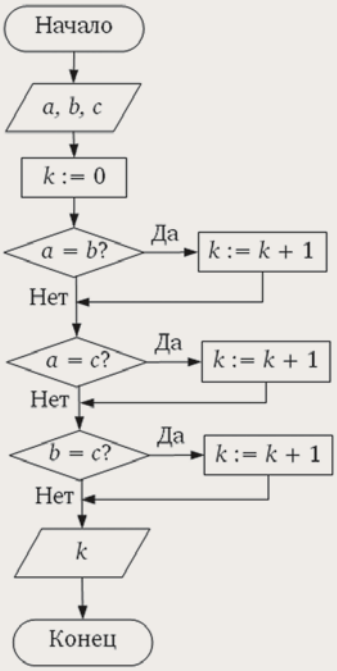


Рис. 52б

Тема для сообщения:
“Разветвляющиеся алгоритмы вокруг нас”

Ветвления и циклы

Ключевые слова:
- ветвление
- цикл
- следование
- алгоритм Евклида
- диалоговая программа

Пример задачи

Любая достаточно сложная программа содержит как циклы, так и ветвления. Пусть Роботу нужно закрасить ряд клеток до клетки, отмеченной буквой Б:

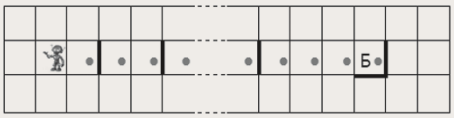


Рис. 53

На его пути в некоторых местах есть стенки, которые можно обойти сверху. Если стенки нет, Робот должен экономить энергию и идти прямо в соседнюю клетку.

Сначала определяем условие остановки Робота в клетке с буквой Б. Там снизу есть стенка, а в других клетках на пути — нет. Поэтому основной цикл можно написать так:

```
нц пока снизу свободно
  ...
кц
```

где многоточие означает какие-то команды.

Что делает Робот на каждом шаге цикла? Он переходит в соседнюю клетку и закрашивает ее. Пере-

ход может быть выполнен двумя способами: если справа свободно, то Робот просто выполняет шаг вправо, а если нет — обходит стенку сверху. Получается такое решение:

```
нц пока снизу свободно
  если справа свободно то
    вправо
  иначе
    вверх; вправо; вниз
  все
  закрасить
кц
```

На блок-схеме (*рис.* 54) хорошо видно, что алгоритм внутри цикла содержит ветвление (оно выделено серым фоном). Кроме того, внутри ветки “да” записана линейная последовательность из трех команд.

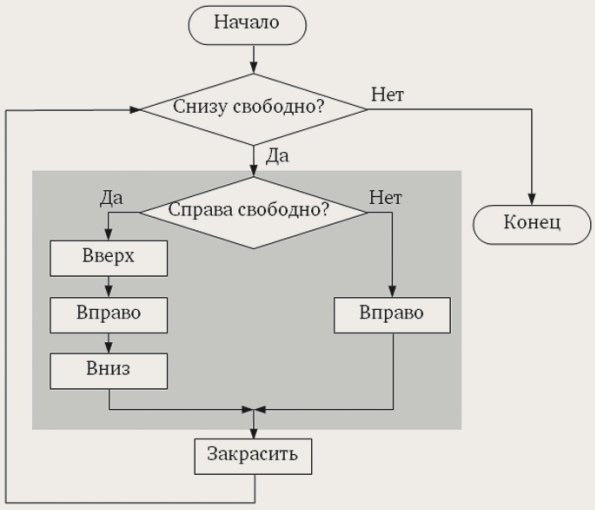


Рис. 54

Исполнитель Раздвоитель

Рассмотрим еще одного исполнителя, которого зовут Раздвоитель. СКИ Раздвоитель содержит две команды:

```
1. вычти 1
2. раздели на 2
```

Можно сказать, что он выполняет обратные действия в сравнении с Удвоителем. Обратите внимание, что команда 2 применима не всегда: ее можно использовать только для тех чисел, которые делятся на 2.

Если Раздвоителю задано натуральное начальное число, то любая из двух команд приводит к уменьшению числа (пока число не станет меньше или равно нулю!). Поэтому он может получить только те числа, которые меньше начального.

Задача. Напишите алгоритм для Раздвоителя, который любое начальное натуральное число превращает в ноль.

Решение. Прежде всего заметим, что таких алгоритмов может быть много. Например, всегда можно просто вычитать единицу (выполнять команду 1), пока не получится ноль:

```
нц пока не ноль
  вычти 1
кц
```

Давайте подумаем, нельзя ли решить задачу быстрее. Чтобы уменьшить время выполнения алгоритма, нужно на каждом шаге уменьшать число как можно быстрее. Команда 2 (**раздели на 2**) быстрее приближает нас к цели, поэтому использовать ее выгоднее, чем команду 1. Но эту команду можно применить не всегда, а только для четных чисел (которые делятся на 2). Следовательно, лучшим решением будет такое:

```
нц пока не ноль
  если четное то
    раздели на 2
  иначе
    вычти 1
  все
кц
```

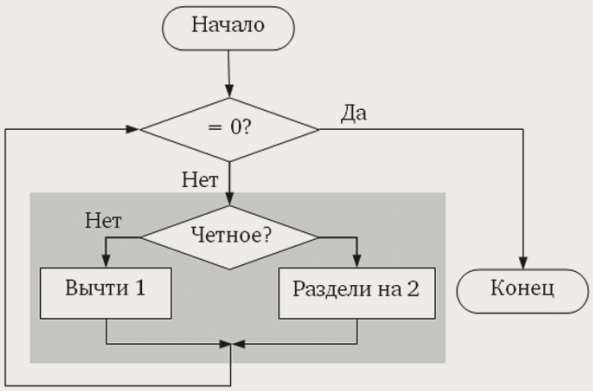


Рис. 55

Этот алгоритм содержит ветвление внутри цикла (оно выделено фоном). Например, для начального числа 20 алгоритм решает задачу не за 20 шагов, как предыдущий, а всего за 6:

Рис. 56

Можно сказать, что второй алгоритм лучше первого, потому что требует меньшего количества действий для получения правильного результата.

Алгоритм Евклида

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

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

Алгоритм Евклида можно записать более строго в виде последовательности шагов:

Вход: натуральные числа a, b.

Шаг 1. Если $a = b$, то перейти к шагу 7.

Шаг 2. Если $a < b$, то перейти к шагу 5.

Шаг 3. Заменить a на $a - b$.

Шаг 4. Перейти к шагу 1.

Шаг 5. Заменить b на $b - a$.

Шаг 6. Перейти к шагу 1.

Шаг 7. Стоп.

Результат: a.

Выполним ручную прокрутку алгоритма: найдем НОД чисел $a = 35$ и $b = 14$. Для этого требуется три шага цикла (то есть команды, заключенные в цикл, — шаги 2–5 — выполняются три раза). На каждом шаге заменяем бóльшее число на разность большего и меньшего:

Действие		a	b
		35	14
a = b?	нет		
a < b?	нет		
a := a - b		21	
a = b?	нет		
a < b?	нет		
a := a - b		7	
a = b?	нет		
a < b?	да		
b := b - a			7
a = b?	да		

После этого алгоритм останавливается, потому что a и b стали равны, условие, проверяемое на шаге 1, истинно, и происходит переход к шагу 7 (на команду Стоп).

Теперь составим блок-схему этого алгоритма. Здесь основная выполняемая операция (шаги 2–6) — это замена бóльшего из двух чисел на их разность. Получается такая схема:

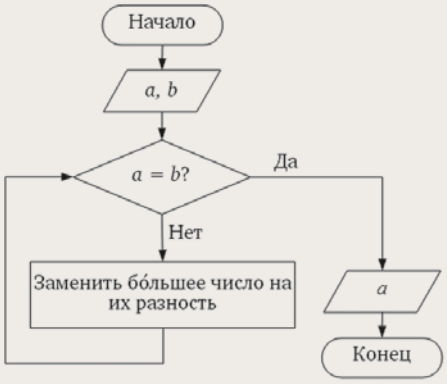


Рис. 57

Согласитесь, что в таком виде алгоритм выглядит понятнее, чем при пошаговом описании.

Обратим внимание, что под блоком “заменить бóльшее число на их разность” скрывается вспомогательный разветвляющийся алгоритм:

Вход: два натуральных числа, a и b.

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

Шаг 2. Заменить a на $a - b$.

Шаг 3. Перейти к шагу 5.

Шаг 4. Заменить b на $b - a$.

Шаг 5. Стоп.

Результат: значения a, b.

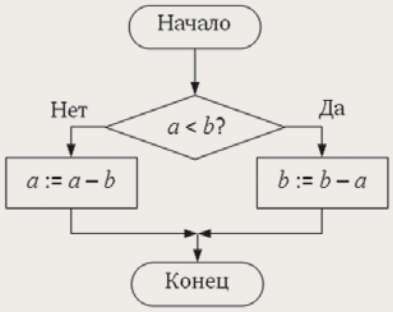


Рис. 58

Блок-схему этого алгоритма можно добавить в основную блок-схему вместо блока “заменить бóльшее число на их разность”:

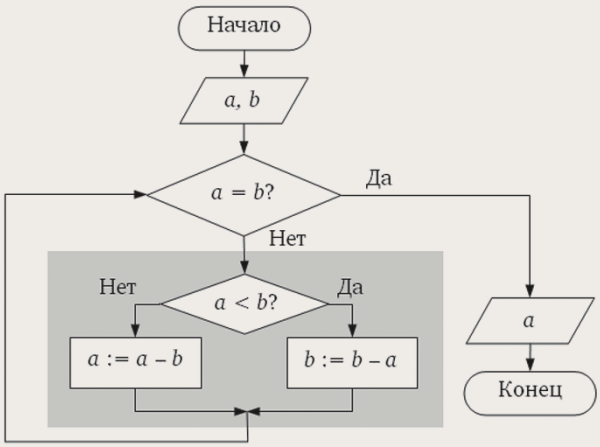


Рис. 59

Таким образом, мы “собрали” всю блок-схему. Обратите внимание, как мы это сделали:

1) сначала выяснили, что алгоритм циклический, и цикл заканчивается при условии $a = b$; это позволило нарисовать схему на *рис.* 57, в которой был один нераскрытый блок — тело цикла;

2) построили отдельно алгоритм для выполнения операций в теле цикла (*рис.* 58);

3) объединили эти два алгоритма в общую блок-схему.

Наверное, вы догадались, что здесь использовался метод проектирования “сверху вниз”, хотя мы не оформляли тело цикла как отдельную процедуру.

Блок-схема на *рис.* 59 описывает достаточно простой алгоритм и уже выглядит довольно запутанно. Поэтому для описания сложных алгоритмов блок-схемы не используют — алгоритм записывают сразу на языке программирования или на *псевдокоде* — смеси естественного языка (русского, английского) и какого-нибудь языка программирова-

ния. Программу на алгоритмическом языке можно записать так:

```
алг Евклид
нач
  цел a, b
  ввод a, b
  нц пока a <> b
    если a < b то
      b := b - a
    иначе
      a := a - b
    все
  кц
  вывод a
кон
```

Она полностью соответствует блок-схеме на *рис.* 59. Условие `a <> b` обозначает "*a* не равно *b*", оно записывается с помощью двух знаков, "<" и ">", без пробела между ними.

В этой программе вы увидели две новые команды школьного алгоритмического языка, **ввод** и **вывод**. Они используются для ввода значений с клавиатуры и вывода результатов на экран, то есть для диалога компьютера с человеком. Такие программы называются *диалоговыми*.

Диалоговая программа — это программа, в которой исходные данные вводятся человеком с клавиатуры, а результаты работы выводятся на экран.

Из чего строятся все алгоритмы?

Как мы увидели, в алгоритмах можно использовать три основных конструкции:

- *следование* (последовательное выполнение, линейный алгоритм);
- *ветвление* (выбор одного из двух вариантов в зависимости от выполнения условия);
- *повторение* (цикл).

В середине XX века было строго доказано, что этих конструкций достаточно для того, чтобы записать любой алгоритм, который только можно придумать.

Алгоритм решения любой задачи можно составить с помощью трех конструкций — **следования**, **ветвления** и **цикла**. Их называют базовыми алгоритмическими структурами.

Контрольные вопросы

1. Найдите в алгоритмах этого параграфа следования, ветвления и циклы.

2. Подумайте, как можно найти НОД двух натуральных чисел, не используя алгоритм Евклида? Какой метод лучше? Почему?

[7] Диалог — это общение двух объектов, например, компьютера и человека.

3. Как вы думаете, в чем достоинства диалоговых программ?

Задачи

1. Придумайте алгоритмы, в которых
а) ветвление находится в теле цикла;
б) цикл находится внутри одной из веток ветвления.

2. Нарисуйте блок-схему алгоритма, который вычисляет остаток от деления одного натурального числа на другое, обрабатывая ошибку "деление на ноль". Запишите этот же алгоритм в пошаговой словесной форме.

3. Напишите программу для Робота, выполнив которую он закрасит все тупики и остановится в конце коридора в клетке Б (длина коридора неизвестна):

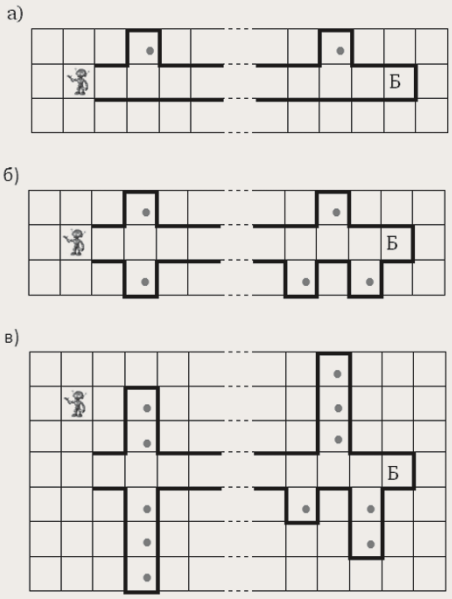


Рис. 60

4. Составьте алгоритм для Раздвоителя, с помощью которого из любого натурального числа $a > 5$ получается число 5 за наименьшее число шагов. Запишите его в словесной форме и в виде блок-схемы.

5. У исполнителя Калькулятор есть команды

```
1. вычти 1
2. раздели на 2
3. раздели на 5
```

Составьте алгоритм для этого исполнителя, с помощью которого из любого натурального числа $a > 3$ получается число 3 за наименьшее число шагов.

6. Напишите программу, которая вычисляет НОД двух целых чисел с помощью алгоритма Евклида. С ее помощью вычислите

а) НОД чисел 64 168 и 82 678; (1234)
б) НОД чисел 358 853 и 691 042; (1111)
в) НОД чисел 6 365 133 и 11 494 962; (171)
г) НОД чисел 17 905 514 и 23 108 855; (3421)
д) НОД чисел 549 868 978 и 298 294 835. (17)

7. Нарисуйте блок-схему улучшенного алгоритма Евклида:

Вход: натуральные числа a, b.

Шаг 1. Если $a = 0$ или $b = 0$, то перейти к шагу 7.

Шаг 2. Если $a < b$, то перейти к шагу 5.

Шаг 3. Заменить a на a mod b (остаток от деления a на b).

Шаг 4. Перейти к шагу 1.

Шаг 5. Заменить b на b mod a (остаток от деления a на b).

Шаг 6. Перейти к шагу 1.

Шаг 7. Стоп.

Результат: $a + b$.

Приведите пример, когда улучшенный алгоритм Евклида работает быстрее алгоритма, приведенного в тексте параграфа.

8. Попробуйте придумать алгоритм, который нельзя записать с помощью циклов, ветвления и следования. Получилось ли у вас?

Тема для сообщения:

“Алгоритм Евклида”

Графические программы

Ключевые слова:

- графический режим
- исполнитель Рисователь
- холст
- пиксель
- координаты
- оси координат
- переменная
- цикл с переменной

Что такое графическая программа?

Много лет назад человек общался с компьютером только в текстовом режиме — вводил с клавиатуры числа и символы, и получал ответ компьютера тоже в форме текста. Мониторы работали в текстовом режиме. Это значит, что весь экран был разбит на прямоугольники-символы (например, 25 строк по 80 символов в каждой) и программа могла управлять каждым символом. При этом было очень сложно что-то нарисовать на экране, потому что рисунок нужно было тоже строить из символов.

Текстовый режим используется и сейчас, потому что в некоторых задачах (например, при настройке компьютера) его возможностей вполне достаточно. Такие программы называются *консольными* (от английского слова *console* — набор устройств для управления компьютером).

Современные мониторы работают в *графическом режиме*. Это значит, что программа может управлять отдельными точками на экране монитора и строить диаграммы, графики, а также выводить на экран изображения, например, фотографии. Конечно, вы уже использовали графические редакторы и создавали рисунки вручную. Теперь мы научимся составлять программы, которые рисуют без нашего участия, автоматически. Для этого мы будем использовать исполнителя Рисователь, который входит в систему КуМир.

Исполнитель Рисователь

Исполнитель Рисователь рисует на *холсте*. Так называется прямоугольная область экрана, доступная для рисования. Если исполнитель попытается нарисовать что-то за пределами холста, эта часть рисунка будет потеряна.

Холст — это прямоугольник, состоящий из отдельных пикселей, то есть *растровый рисунок*. Каждый пиксель имеет две координаты:

x — расстояние от пикселя до левой границы холста;

y — расстояние от пикселя до верхней границы холста.

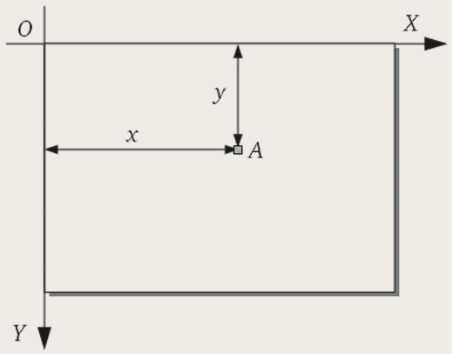


Рис. 61

Можно сказать, что Рисователь использует прямоугольную систему координат, в которой ось X направлена вдоль верхней границы холста вправо, а ось Y — вдоль нижней границы холста **вниз**. Обратите внимание на непривычное направление оси Y — в математике она обычно направлена вертикально вверх.

Пиксель в левом верхнем углу холста имеет координаты (0,0). Нумерация с нуля часто используется в программировании. В нашем случае это значит, что пиксель находится одновременно на левой границе и на верхней границе холста (оба расстояния до границ равны нулю).

Напишем первую программу для Рисователя: создадим холст размером 500 × 400 пикселей (ширина — 500 пикселей, высота — 400 пикселей) и зальем его белым цветом:

```
использовать Рисователь
алг Холст
нач
  новый лист( 500, 400, белый )
кон
```

В первой строке мы подключаем исполнителя Рисователь, а единственная строка основной программы создает нужный холст, вызывая команду **новый лист**, входящую в СКИ Рисователя:

```
новый лист( 500, 400, белый )
```

Первое число означает ширину холста в пикселях, второе — высоту, а третье — название цвета, которым заливается холст.

Вот все простые цвета, которые можно использовать в КуМире:

белый, **черный**, **серый**, **фиолетовый**, **синий**, **голубой**, **зеленый**, **желтый**, **оранжевый**, **красный**.

Управление пикселями

Мы уже говорили, что в графическом режиме программа может управлять отдельно каждым пикселем. Этого достаточно, чтобы нарисовать любой рисунок, даже самый сложный.

У Рисователя есть команда **пиксель**, которая изменяет цвет пикселя с заданными координатами. Например, закрасить синим цветом пиксель с координатами[7] (10,20) можно так:

```
пиксель( 10, 20, синий )
```

Данные в скобках, которые передаются процедуре **пиксель**, называются *аргументами*. Первые два аргумента — это *x*-координата и *y*-координата пикселя, а третий — нужный цвет.

Теперь нарисуем горизонтальную синюю линию из точки (10,20) в точку (15,20). Линия на рисунке состоит из отдельных пикселей, поэтому нужно закрасить синим цветом пиксели с координатами (10,20), (11,20), (12,20), (13,20), (14,20) и (15,20). Программа может выглядеть так:

```
пиксель( 10, 20, синий )
пиксель( 11, 20, синий )
пиксель( 12, 20, синий )
пиксель( 13, 20, синий )
пиксель( 14, 20, синий )
пиксель( 15, 20, синий )
```

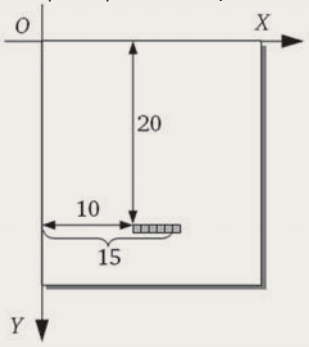


Рис. 62

Чтобы не писать много похожих команд, мы попытаемся построить цикл. Заметим, что в этих командах различается только первая координата (она выделена фоном). То есть это *переменная* величина (вспомните, что такое переменная в программировании). Обозначим первую координату пикселя именем **x**. Тогда нам нужно шесть раз выполнить команду

```
пиксель( x, 20, синий )
```

изменяя значение переменной **x** от 10 до 15. Мы уже можем сделать это так (вспомните цикл *N* раз):

```
цел x
x := 10
нц 6 раз
  пиксель( x, 20, синий )
  x := x + 1
кц
```

Здесь введена целая переменная **x**, которой сначала присваивается значение 10. Затем шесть раз выполняется команда, закрашивающая синим цветом пиксель с координатами (**x**, 20), и каждый раз значение **x** увеличивается на единицу.

В алгоритмическом языке есть еще один вид цикла — *цикл с переменной*, — который позволяет записать то же самое задание для Рисователя проще:

```
цел x
нц для x от 10 до 15
  пиксель( x, 20, синий )
кц
```

В заголовке цикла мы сказали, что тело цикла нужно выполнить для всех целых значений **x** от 10 до 15 включительно. Всю работу по увеличению переменной **x** на каждом шаге цикла программа берет на себя, нам не нужно об этом заботиться.

Итак, мы увидели, что с помощью команды **пиксель** можно достаточно легко нарисовать горизонтальную линию. Это связано с тем, что одна из координат (здесь — *y*-координата) не изменяется, она одинакова для всех точек линии. Вертикальная линия рисуется так же просто (не меняется *x*-координата).

А что если нужно нарисовать наклонную линию, например, соединить линией точки (10,20) и (20,50)? Здесь будут изменяться обе координаты, причем *y*-координата меняется быстрее, чем *x*-координата (подумайте, почему?). Поэтому решить такую задачу, используя только команду **пиксель**, очень непросто (можете попробовать). Еще сложнее нарисовать окружность.

Чтобы облегчить работу программистам, в СКИ графических исполнителей добавляют команды, которые рисуют сразу целые геометрические фигуры: линии, прямоугольники, окружности и др. Они называются *графическими примитивами*, используемыми в векторной графике. В следующем параграфе мы познакомимся с графическими примитивами в СКИ Рисователя.

Контрольные вопросы

1. Чем отличаются программы, работающие в текстовом и графическом режимах?
2. Что такое консольная программа?
3. Как определяются координаты пикселя на холсте? Чем необычна эта система координат?

[7] Координаты пикселей записываются в круглых скобках через запятую, сначала — *x*-координата, потом — *y*-координата.

Как вы думаете, почему в компьютерных программах используется именно такая система координат?

4. Как создать новый холст размером 200 × 100 пикселей и залить его синим цветом?

5. Что будет, если мы вызовем команду **пиксель** так:

```
пиксель( синий, 10, 20 )
```

В чем ошибка?

6. Чем отличается цикл с переменной от цикла **N раз** и цикла с условием?

7. Можно ли зациклить программу, используя цикл с переменной?

8. Почему сложно рисовать наклонные линии и окружности, используя только команду **пиксель**?

Задачи

1. Определите координаты всех пикселей на рисунке:

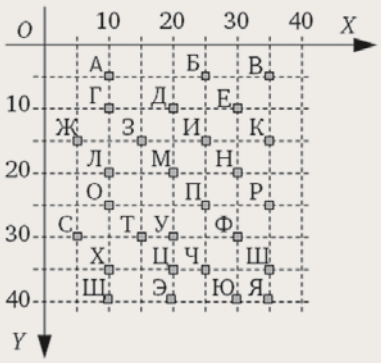


Рис. 63

2. Нарисуйте горизонтальную и вертикальную линии с помощью команды **пиксель** и циклов.

3. Нарисуйте прямоугольник размером 100 × 50 пикселей с помощью команды **пиксель** и циклов.

Окончание в следующем номере

ИНТЕЛЛЕКТ БУДУЩЕГО
малая академия наук

Всероссийский конкурс проектных работ
«Созидание и творчество»

Работы принимаются в феврале – марте 2015 г.

На конкурс принимаются учебные проекты по всем школьным предметам. Продуктом проекта могут быть: презентации к уроку, учебные пособия, дидактические материалы, описание учебной игры, учебные сайты и т.п.

По итогам автору проекта присваивается звание Лауреата (+ диплом I, II или III степени) либо статус Участника (+ свидетельство). Для научных руководителей – сертификаты.

Подробно: unk.future4you.ru (раздел «Созидание и творчество»).
Задать вопрос kp@future.org.ru

Публикация на сайте
vd.future4you.ru

Всероссийская открытая интернет-выставка
«Достижения учащихся»

Принимаются учебные, экологические, социальные, научные и другие проекты. Все участники получают свидетельства, приглашаются на турнир на Чёрном море.

Всероссийские конкурсы и предметные олимпиады
на сайте www.future4you.ru