# Часть I

# АЛГОРИТМЫ (ОБЛЕГЧЕННЫЙ МАТЕРИАЛ ДЛЯ ПЕРВОГО ЗНАКОМСТВА)

# Глава 1

# АЛГОРИТМЫ – ЭТО ОЧЕНЬ ПРОСТО!

## §1. ПЕРВОЕ ЗНАКОМСТВО С АЛГОРИТМОМ

Предположим, мать говорит сыну: «Сходи в булочную. Купи батон белого и батон черного». Слова матери – алгоритм, показанный на рис. 1. Данный алгоритм нельзя признать удачным. Он не отвечает на вопрос: что делать, если в магазине нет хлеба.

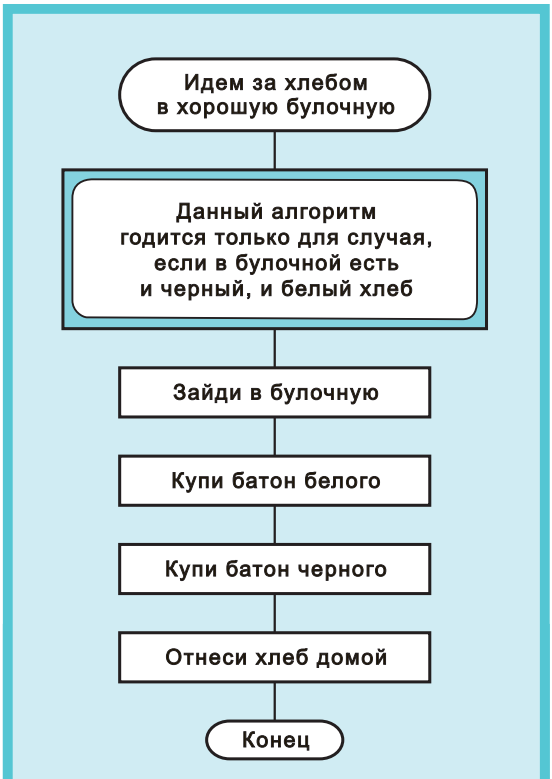



Рис. 1. Алгоритм «Идем за хлебом в хорошую булочную»

Чтобы исправить оплошность, нужно придумать другой алгоритм, который учитывает реальные условия. Предположим, в булочной черный хлеб водится всегда, а с белым случаются перебои. В таком случае мать могла бы сказать: «Купи батон белого и батон черного. Если белого не будет, возьми два черного». Словесная формулировка этого алгоритма показана на рис. 2, а блок-схема – на рис. 3.

К последнему алгоритму тоже можно придраться. Как быть, если в продаже нет черного хлеба? Чтобы учесть все варианты, мать должна дать сыну более сложную инструкцию: «Сходи за хлебом. Один батон белого и один черного.

Белого не будет – купи два черного. И наоборот. А если никакого нет, обойдемся».

Соответствующая блок-схема показана на рис. 4.

**Название алгоритма**

Идем за хлебом в обычную булочную

**Комментарий**

Данный алгоритм годится для двух случаев:
- в продаже есть и черный, и белый хлеб;
- в продаже есть только черный.

**Команды**

- Зайди в булочную
- Узнай, есть ли в продаже белый хлеб
- Если белый есть, купи батон белого и батон черного
- Если белого нет, купи два батона черного
- Отнеси хлеб домой
- Конец

Рис. 2. Словесное описание алгоритма

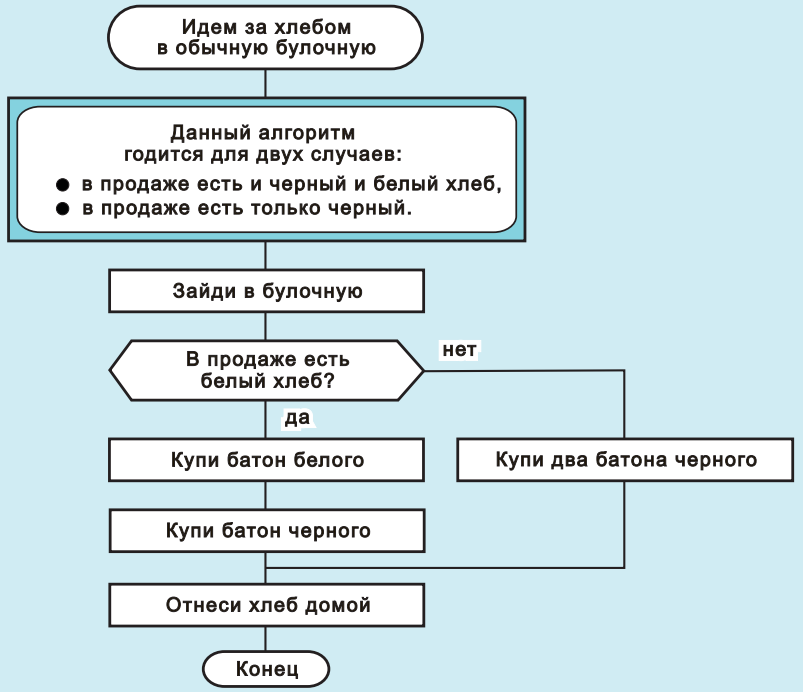



Рис. 3. Алгоритм «Идем за хлебом в обычную булочную». Словесное описание этого алгоритма дано на рис. 2

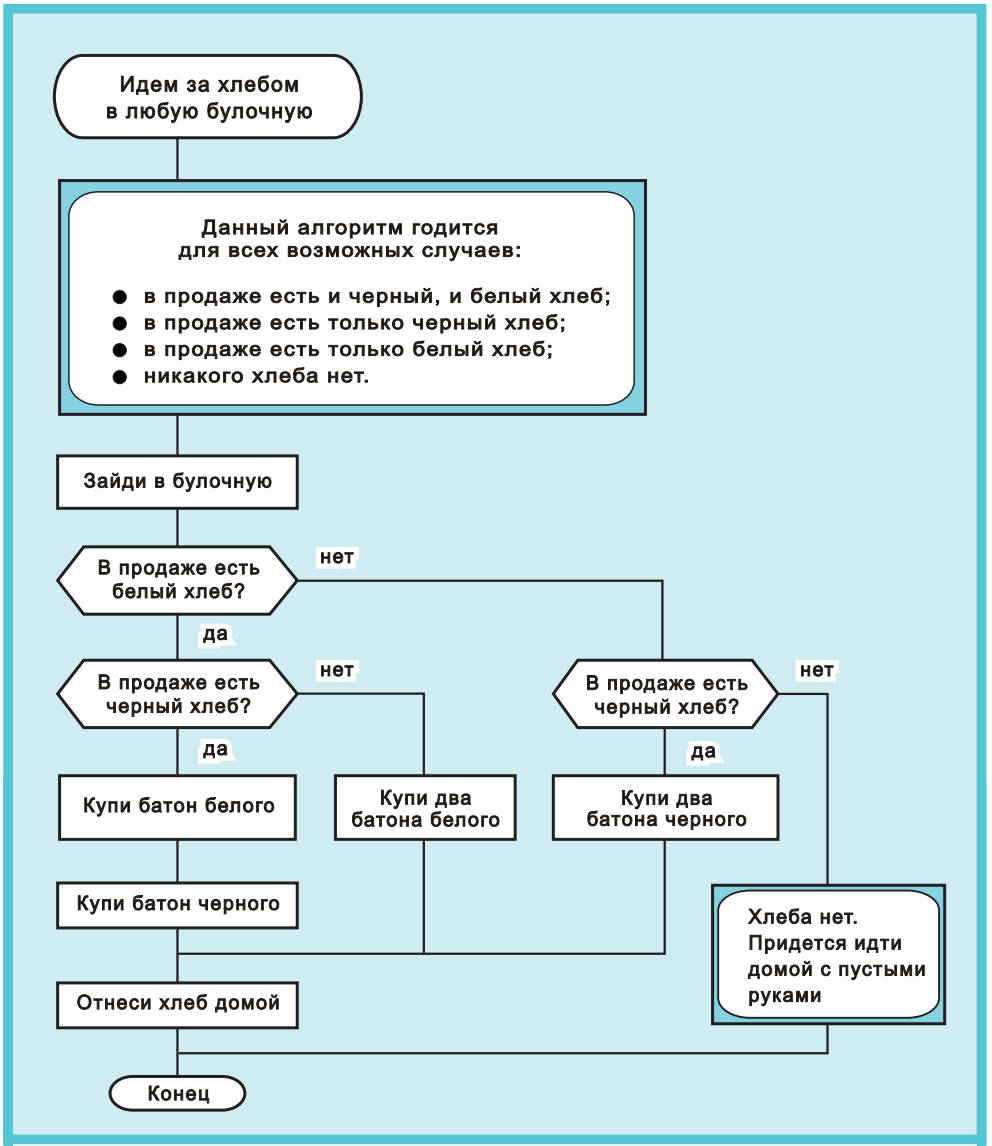



Рис. 4. Алгоритм «Идем за хлебом в любую булочную»

## §2. СРАВНЕНИЕ АЛГОРИТМОВ

Взглянем на три алгоритма на рис. 1, 3, 4. Какой из них следует считать хорошим, а какой – плохим? Алгоритм считается хорошим, если он правильно работает при всех реальных условиях. В противном случае алгоритм считается плохим.

Применим это правило к нашим алгоритмам. Легко видеть, что только один алгоритм (рис. 4) будет работать при любых условиях.

В самом деле, если в булочной есть и белый, и черный хлеб, воображаемый «бегунок» побежит от начала к концу алгоритма по левой вертикальной линии.

Если белый есть, а черного нет, бегунок помчится чуть правее – через икону «Купи два батона белого».

Если белого нет, а черный есть, бегунок побежит еще правее – через икону «Купи два батона черного».

Наконец, если в булочной вообще нет хлеба, бегунок выберет крайний правый путь – через икону «Хлеба нет. Придется идти домой с пустыми руками».

Что же мы узнали? Алгоритм на рис. 4 – хороший алгоритм, так как он выполняет свою работу *при любых условиях*. И наоборот, алгоритмы на рис. 1 и 3 – это плохие, «неполноценные» алгоритмы.

Можно сказать и по-другому. Алгоритм на рис. 4 – правильный, безошибочный, работоспособный алгоритм. А схемы на рис. 1 и 3 не отвечают на вопрос: что делать, если в булочной нет хлеба. Значит, эти алгоритмы неработоспособны; они содержат ошибку. На рис. 4 ошибка исправлена.

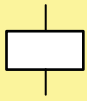
## §3. ЧТО ТАКОЕ ДРАКОН-СХЕМЫ? ЗАЧЕМ НУЖНЫ ИКОНЫ?

Блок-схемы, нарисованные по правилам языка ДРАКОН, называются *дракон-схемы*. Они состоят из простых рамок – *икон*, соединенных между собой. На рис. 1 таких икон семь. Верхняя называется «заголовок». Та, что пониже – «комментарий». Следующие четыре – «действие». А самая нижняя – «конец».

Что такое «Заголовок»

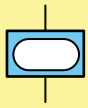
Овальная икона, расположенная в самом начале схемы. Внутри нее пишут название алгоритма

Что такое «Действие»



Прямоугольная икона, внутри которой пишут команды алгоритма

Что такое «Комментарий»



Прямоугольная икона с двойной рамкой, внутри которой пишут не команды, а пояснения

Что такое «Конец»

Маленькая овальная икона, расположенная в самом конце алгоритма. Внутри нее пишут слово «Конец»

## §4. АЛГОРИТМ

Каждый человек, находясь на работе, ведет себя отнюдь не случайно. Он стремится к четко определенной цели и выполняет нужные для этого действия. Таким образом, любую работу (за исключением сложных творческих операций) можно рассматривать как хорошо определенный каскад действий, то есть алгоритм. Нетрудно заметить, что алгоритмы окружают нас повсюду.

Алгоритм – последовательность шагов, ведущих к цели. Можно сказать и по-другому. Алгоритм – последовательность команд, помогающих решить задачу.

Что такое алгоритм

Это последовательность действий, ведущих к поставленной цели

Команды алгоритма можно записать по-разному, например:

- в виде дракон-схемы;
- в виде текста.

Чтобы убедиться в этом, взгляните на рис. 5.

Рис. 5. Преврати текст алгоритма в дракон-схему

## §5. КАЖДАЯ КОМАНДА АЛГОРИТМА ДОЛЖНА ВЫПОЛНЯТЬСЯ ТОЧНО И ОДНОЗНАЧНО

Рассмотрим алгоритм на рис. 6. Предположим, алгоритм выполняет робот.

Робот – типичный буквоед. Обычно педантов и буквоедов не любят. Но в мире алгоритмов свои законы. Там буквоеды в большом почете. Чтобы робот понял алгоритм, каждая команда должна быть написана «по-буквоедски». Точно и однозначно.

На рис. 6 алгоритм написан именно так – точно и однозначно. Там ясно сказано – надо купить килограмм картошки и кочан капусты. Это значит, что любые другие покупки запрещены.

А что мы видим на рис. 7? Тут ситуация иная. Команда «Купи продукты» – плохая, неоднозначная команда. Ее невозможно исполнить. Потому что нигде не сказано, какие именно продукты нужно купить.

Такие умолчания в алгоритмах недопустимы. Они рассматриваются, как ошибки.

Выполните упражнение на рис. 8.

## §6. ИКОНА «ВОПРОС»

В верхней части рис. 9 изображена икона «вопрос». Она называется так потому, что внутри нее пишут «да-нетный вопрос». То есть вопрос, на который можно ответить либо «да», либо «нет». Все другие ответы запрещены.

**Рис. 6. Хороший алгоритм. Здесь ясно сказано: нужно купить килограмм картошки и кочан капусты**

**Рис. 7. Плохой алгоритм. Его нельзя выполнить, так как неизвестно, какие именно продукты нужно купить**

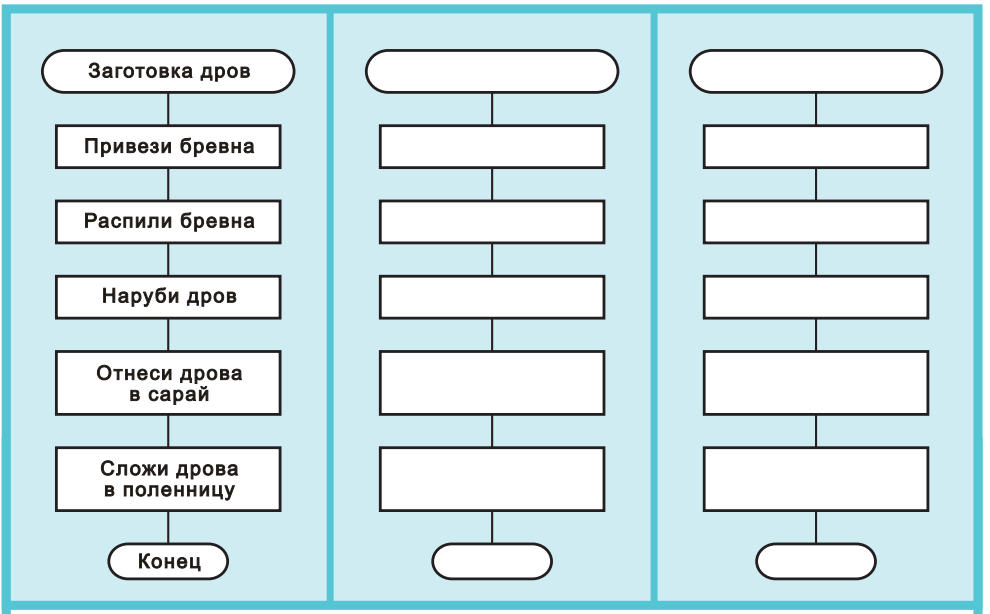



Рис. 8. Придумай два алгоритма по образцу, показанному слева

Вот примеры да-нетных вопросов: утюг сломался? Вася купил хлеб? Поезд пришел? Преступника арестовали? Тетя приехала? «Спартак» выиграл? Эта лужа больше, чем та? На улице температура выше нуля?

Икона «вопрос» имеет один вход сверху и два выхода: вниз и вправо. Выход влево запрещен и никогда не используется.

Бывают случаи, когда нужно выбрать одно действие из двух. В этом случае удобно использовать икону «вопрос». При ответе «да» выполняется одно действие, при «нет» – другое. Икона «вопрос» нужна, чтобы сделать в алгоритме развилку. В этом легко убедиться, взглянув на рис. 9.

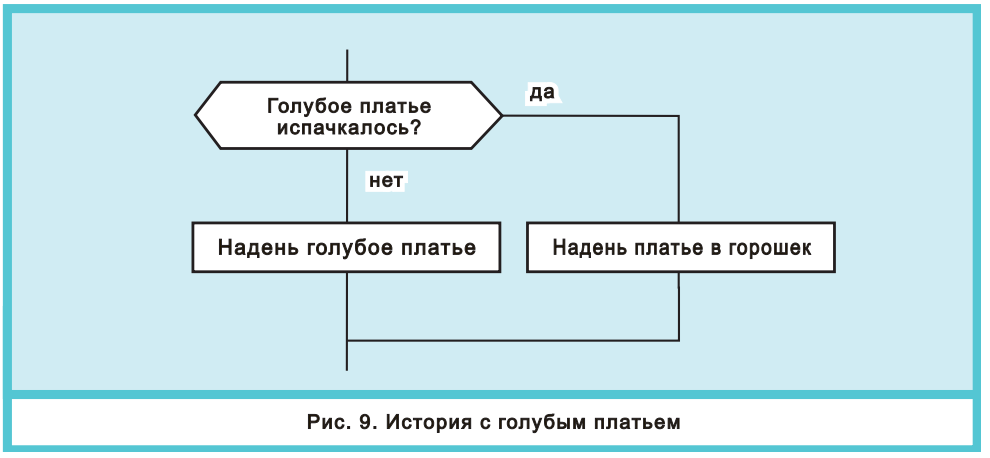
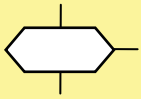



Рис. 9. История с голубым платьем

Иногда в алгоритме надо сделать не одну, а много развилок. Для этого используют несколько икон «Вопрос» (рис. 4).

Что такое «Вопрос»



- Ромб, у которого отрезаны верхний и нижний углы.
- Внутри фигуры пишут да-нетный вопрос, то есть вопрос, на который можно ответить либо «да», либо «нет».

## §6. ВЫВОДЫ

1. Умение выразить свою мысль в виде алгоритма имеет много преимуществ. Такое умение пригодится почти каждому – ведь оно делает интеллект более гибким и эффективным.
2. Алгоритмы играют огромную роль в жизни общества. Они оказывают заметное влияние на эффективность экономики, уровень жизни, а также на умственное развитие цивилизации.

# Глава 2

# АЛГОРИТМЫ И ПРОЦЕДУРНЫЕ ЗНАНИЯ

## §1. ПРОЦЕДУРНЫЕ И ДЕКЛАРАТИВНЫЕ ТЕКСТЫ

Давайте совершим экскурсию в библиотеку. На полках стоят тысячи книг. Толстых и тонких. Художественных и научных. Всяких.

Все тексты в книгах можно разделить на две части:

- процедурные,
- декларативные.

Чтобы уяснить, чем они различаются, обратимся к таблице 1.

***Таблица 1***

| Это процедурный текст | Это декларативный текст |
|---|---|
| Скинь мантилью, ангел милый,<br>И явись, как яркий день!<br>Сквозь чугунные перилы<br>Ножку дивную продень! | По синим волнам океана,<br>Лишь звезды блеснут в небесах,<br>Корабль одинокий несется,<br>Несется на всех парусах. |

Текст в левом столбце при желании можно превратить в набор команд:

1. Скинь мантилью.
2. Явись, как ясный день.
3. Продень ножку сквозь перила.

Текст в правом столбце такому преобразованию не поддается. Его невозможно превратить в команды.

Рассмотрим еще один пример (табл. 2).

***Таблица 2***

| Это процедурный текст | Это декларативный текст |
|---|---|
| Сходи к бабушке, возьми у нее телевизор и отнеси в ремонт. | Телевизор был старый, покрытый толстым слоем пыли. Видимо, им уже давно не пользовались. |

Левый текст без труда превращается в команды:

1. Сходи к бабушке.
2. Возьми у нее телевизор.
3. Отнеси его в ремонт.

Правый текст имеет принципиально иную логическую структуру. Как ни старайся, как ни хитри, а в команды его не превратишь!

Если текст содержит достоверные и обоснованные сведения, такие сведения называются знанием.

**Что такое процедурные знания**

Это знания о действиях, планах, процессах и инструкциях, которые можно легко превратить в последовательность команд

**Что такое декларативные знания**

Это описания различных фактов и явлений, которые невозможно превратить в последовательность команд

## §2. КАК ПРЕВРАТИТЬ ПРОЦЕДУРНОЕ ЗНАНИЕ В АЛГОРИТМ?

В верхней части рис. 10 находится текст, содержащий процедурное знание. Это знание можно превратить в дракон-схему (алгоритм).

Пробежим взглядом по схеме сверху вниз. Мы обнаружим, что алгоритм нарисован не хаотично, а упорядоченно. Тут действует

> ***Правило.*** **Чем ниже нарисована икона, тем позже выполняется записанная в ней команда. (Для краткости: «Чем ниже – тем позже»).**

Данное правило выполняется во всех алгоритмах.

Здесь нас поджидает сюрприз. Переведем взгляд со схемы на исходный текст. Налицо разительный контраст – в тексте данное правило не соблюдается! Таким образом, алгоритм сильно отличается от исходного текста. В чем отличие?

А вот в чем – порядок записи команд коренным образом изменился. Это хорошо видно на рис. 11.

- Первая команда текста (*Сходи к бабушке*) в алгоритме стала не первой, а второй.
- Вторая команда текста (*возьми у нее телевизор*) переехала совсем в другое место.

- Третья команда текста (*и отнеси его в ремонт*) стала предпоследней.
- Четвертая команда текста (*На случай, если бабушки не будет дома*) сохранила свое место – осталась четвертой.
- И наконец, самое удивительное! Последняя команда текста (*возьми с собой ключ от ее квартиры*) перескочила из конца в начало. В алгоритме она стала самой первой.

Как можно объяснить эти чудесные превращения? Ответ довольно прост.

Исходный текст, представленный на рис. 10 и 11, написан на естественном языке. В этом языке закон «Чем ниже – тем позже» не действует. А в алгоритме ситуация обратная. Там этот закон – святая святых и обязательно исполняется.

Отсюда вытекает

**Правило**
**преобразования процедурного текста в алгоритм**

Чтобы превратить процедурное знание, записанное на естественном языке, в дракон-схему, необходимо:

- расчленить исходный текст на фрагменты, чтобы в каждом фрагменте содержалась одна команда;
- записать команды в виде упорядоченной последовательности согласно принципу «Чем ниже – тем позже»;
- добавить иконы «заголовок» и «конец»;
- при необходимости внести дополнительные команды, устраняющие пробелы исходного текста и превращающие дракон-схему в логически законченный алгоритмический рассказ.

*Примечание.* На рис. 10 и 11 дополнительными командами являются:

- Позвони в дверь.
- Открой дверь своим ключом.
- Войди в квартиру.

## §3. МАТЕМАТИЧЕСКИЕ И «БЫТОВЫЕ» АЛГОРИТМЫ

Во многих учебниках описаны математические алгоритмы. Это хорошо, но мало. Разумеется, студенты должны назубок знать математику. Но в жизни бывают неожиданности. Некоторые студенты, овладев математикой, затрудняются составлять алгоритмы, в которых отсутствуют математические формулы.

Чтобы избежать подобных неприятностей, студенты должны уметь распознавать «нематематические» алгоритмы, которые постоянно встречаются в жизни.

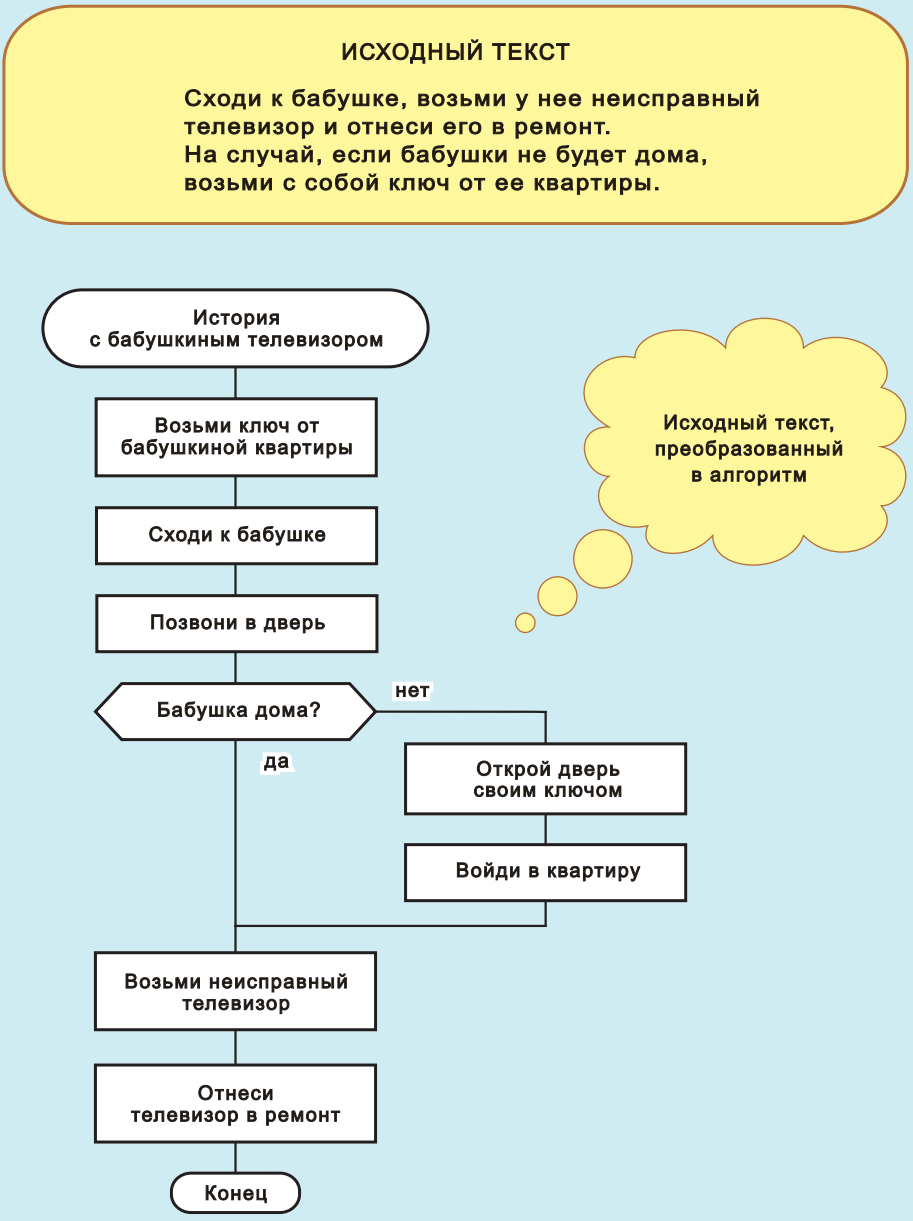



Рис. 10. Процедурное знание, представленное в двух формах:
- в виде обычного текста,
- в виде алгоритма

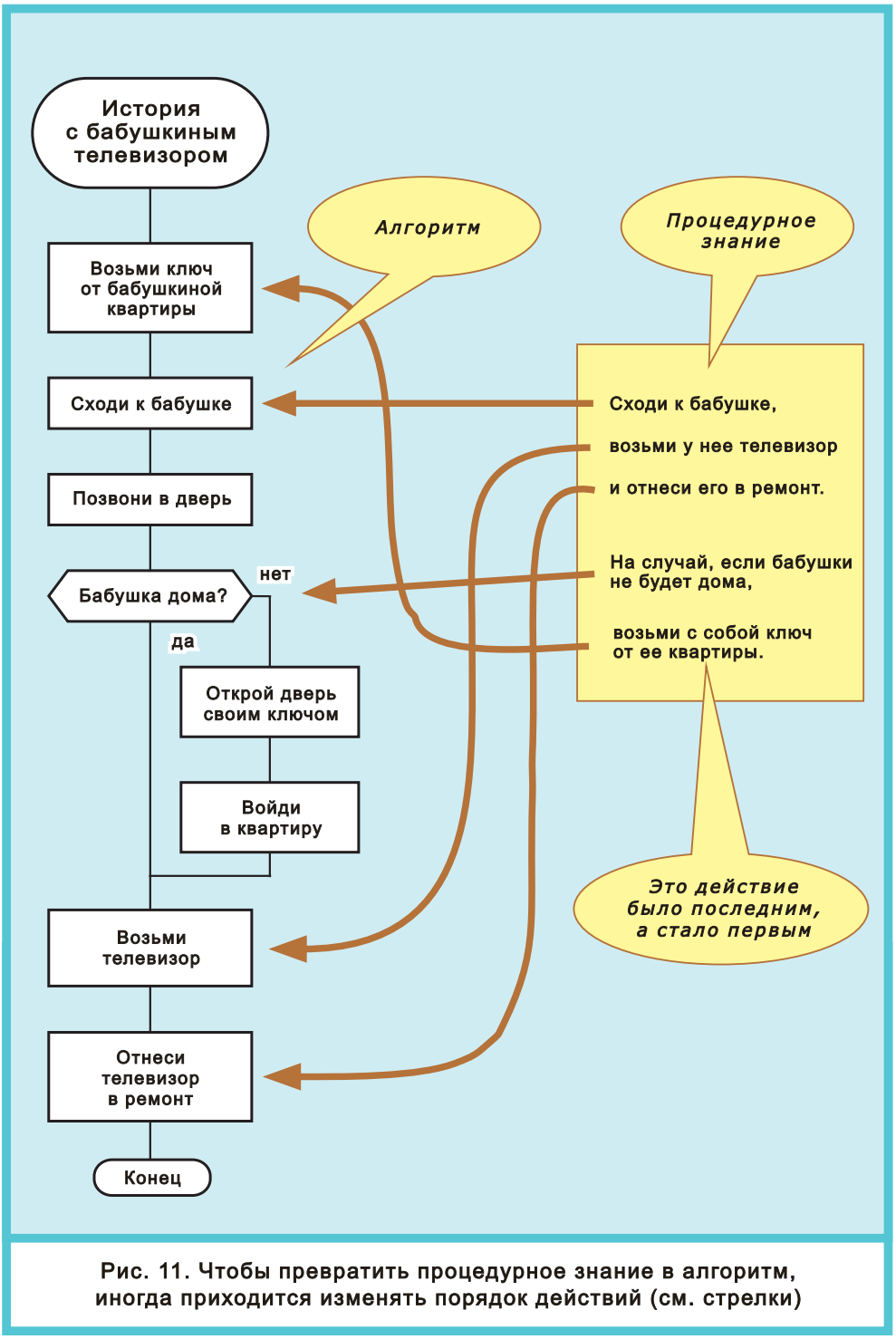



Рис. 11. Чтобы превратить процедурное знание в алгоритм, иногда приходится изменять порядок действий (см. стрелки)

Такие алгоритмы можно условно назвать «бытовыми». Приведем пример. Перед нами стихотворение Генриха Гейне.

Вставай, слуга! Коня седлай!
Чрез рощи и поля
Скачи скорее ко двору
Дункана-короля.

Зайди в конюшню там и жди.
И, если кто войдет,
Спроси: «Которую Дункан
Дочь замуж выдает?»

Коль чернобровую – лети
Во весь опор назад.
Коль ту, что с русою косой,
Спешить не надо, брат.

Тогда ступай на рынок ты,
Купи веревку там,
Вернися шагом и молчи –
Я догадаюсь сам.

Это стихотворение легко превращается в алгоритм (рис. 12). Там же виден кусочек декларативного знания, которое записано в иконе «комментарий» – («Я догадаюсь сам»).

## §4. ЧТО ТАКОЕ ПЕРЕКЛЮЧАТЕЛЬ?

Схема на рис. 13 позволяет сделать в алгоритме развилку на три направления. Для этого используются две иконы «вопрос». Однако задачу можно решить и по-другому – с помощью переключателя (рис. 14).

*Переключатель* – разветвление алгоритма на несколько тропинок, которые потом сливаются в одну. Переключатель – сложная структура. Она строится из простых кирпичиков. Такими кирпичиками служат иконы «выбор» и «вариант». Зачем они нужны?

Икона «выбор» содержит вопрос, имеющий несколько ответов. Каждый ответ пишут в отдельной рамочке – иконе «вариант».

***Пример 1.*** На рис. 12 в иконе «вариант» записан вопрос: «Какую дочь Дункан выдает замуж?». На этот вопрос есть два ответа, записанные в иконах «вариант»:

- чернобровую;
- с русою косой.

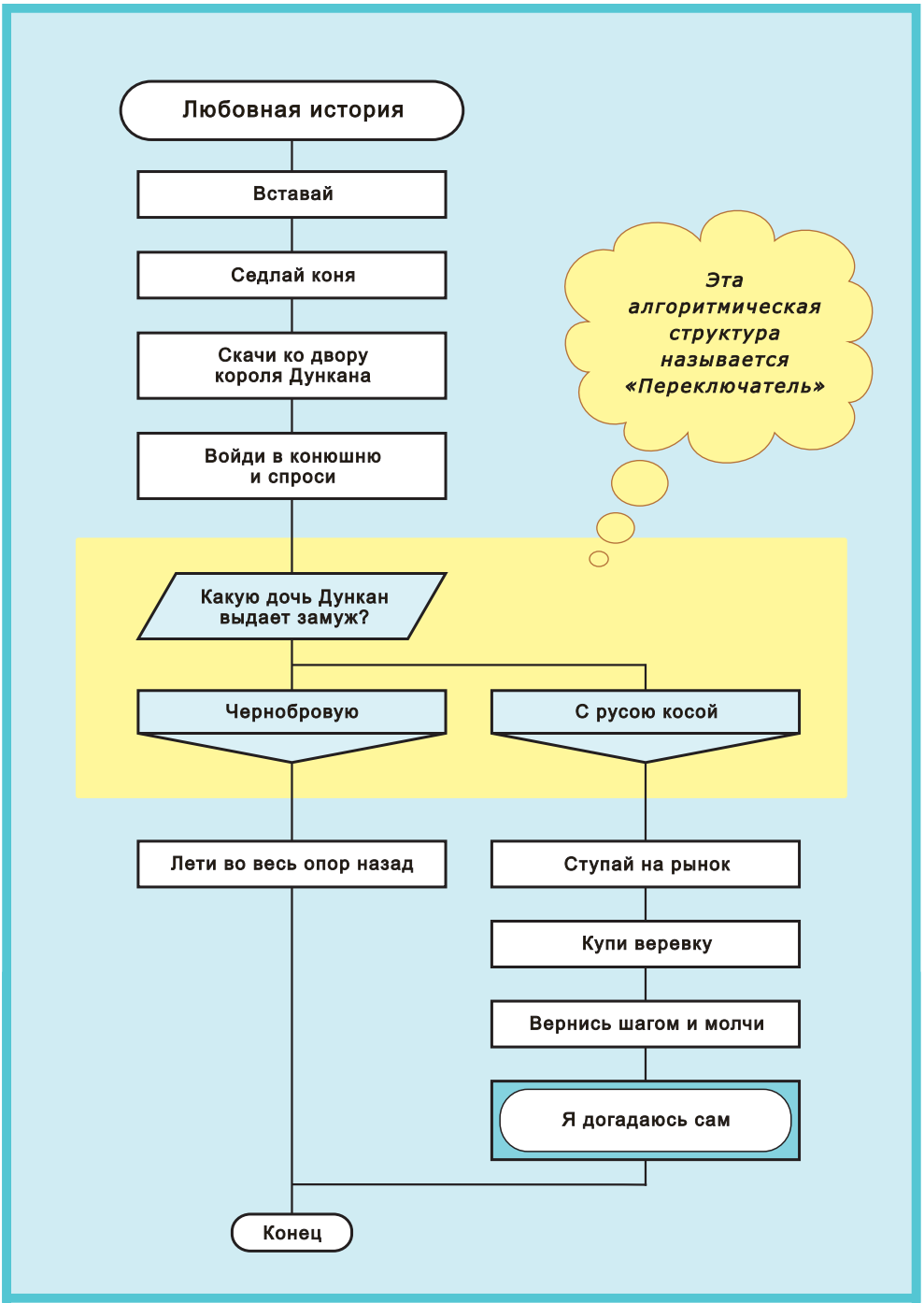



Рис. 12. Стихотворение Гейне, преобразованное в алгоритм

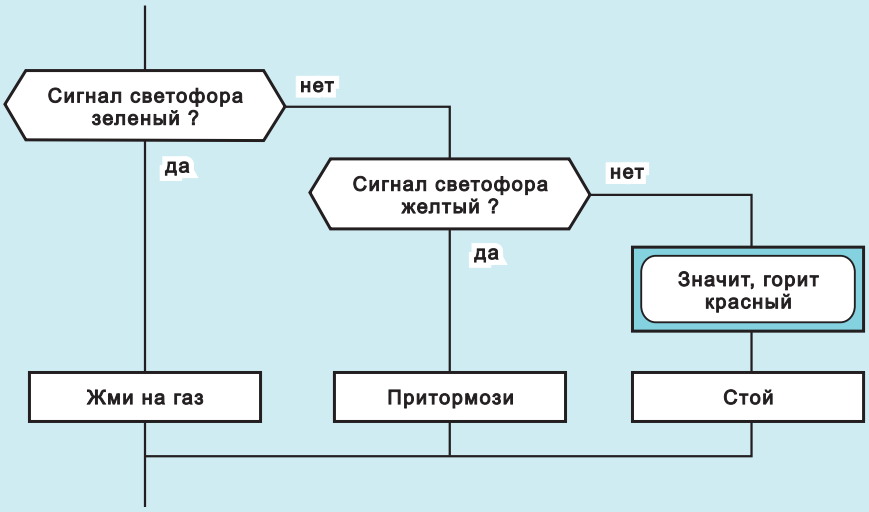



Рис. 13. Развилка на три направления, построенная с помощью иконы «Вопрос»

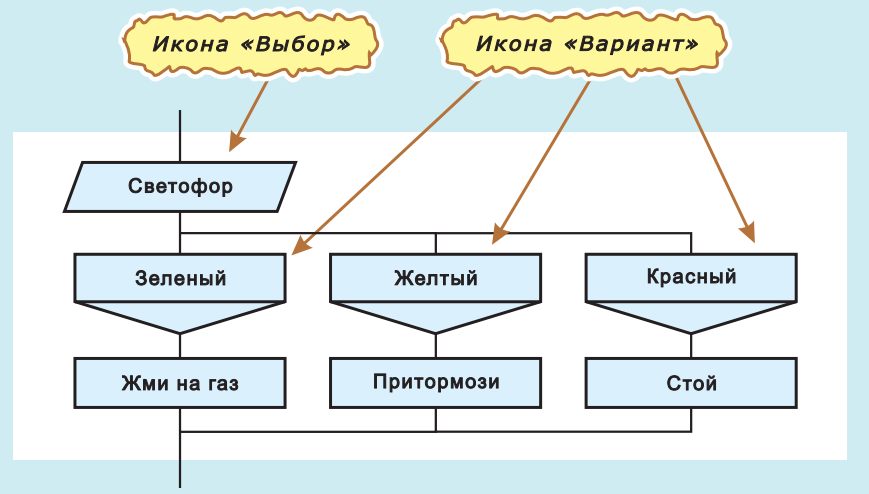



Рис. 14. Развилка на три направления, построенная с помощью переключателя

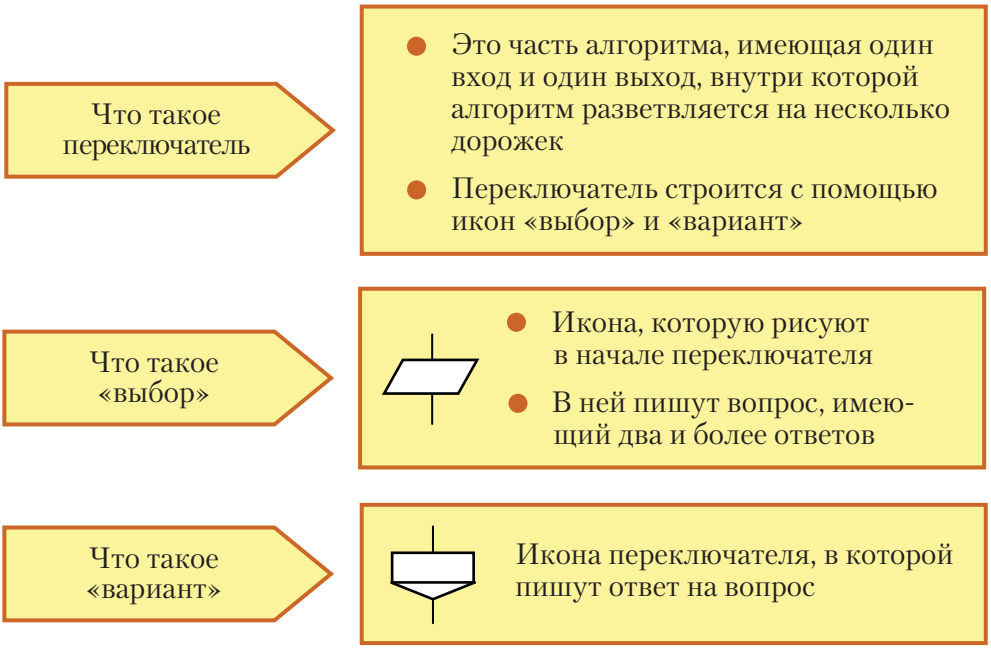
***Пример 2.*** На рис. 14 картина иная. На первый взгляд там нет вопроса. Однако на самом деле вопрос есть, правда неявный. Чтобы убедиться, слово «светофор» прочитаем так: «Какой сигнал светофора сейчас горит?». На этот вопрос есть три ответа:

- зеленый,
- желтый,
- красный.

Попробуем описать, как работает переключатель. Описание всегда начинается со слова «если». Вот примеры.

- Если Дункан выдает замуж чернобровую дочь – лети во весь опор назад.
- Если Дункан выдает замуж дочь с русою косой – ступай на рынок, купи веревку, вернись шагом и молчи (рис. 12).
- Если светофор зеленый – жми на газ.
- Если светофор желтый – притормози.
- Если светофор красный – стой (рис. 14).

Дополнительный пример показан на рис. 15.




## §5. БЛОК-СХЕМЫ АЛГОРИТМОВ

*Блок-схема* – чертеж алгоритма, описывающий последовательность действий с помощью графических фигур и соединительных линий. Это поучительная часть истории алгоритмов.

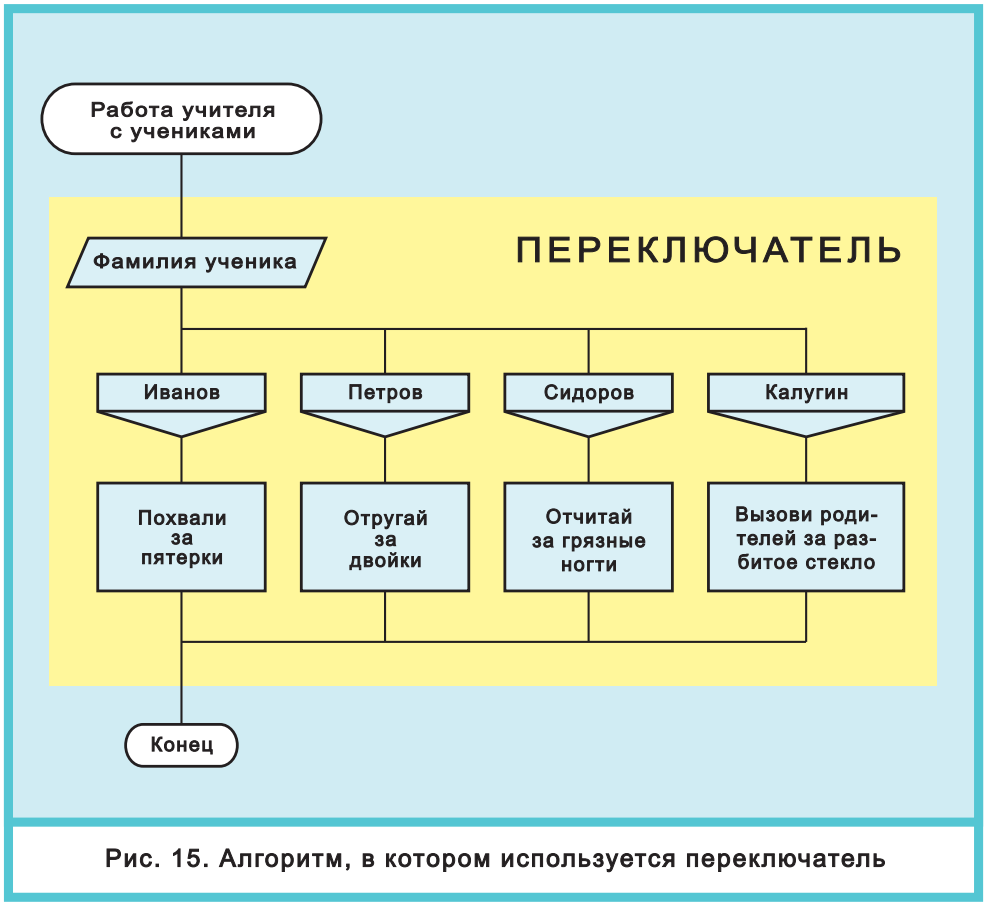



**Рис. 15. Алгоритм, в котором используется переключатель**

На рис. 16 показаны наиболее употребительные фигуры, используемые при вычерчивании блок-схем. Они соответствуют международному стандарту ISO 5807–85 и отечественному ГОСТ 19.701-90.

Хотя стандарты на блок-схемы считаются действующими, фактически они давно устарели.

С появлением дракон-схем блок-схемы полностью потеряли свое значение, так как они во всех отношениях уступают дракон-схемам. (Подробнее см. главу 15).

## §6. АНАЛОГИ ДРАКОН-СХЕМ

Аналогами дракон-схем являются блок-схемы алгоритмов, диаграммы Насси-Шнейдермана, псевдокод (язык описания алгоритмов) и др. Другим аналогом дракон-схем служат диаграммы поведения (*behaviour diagrams*)

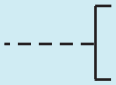
| Фигура | Название фигуры | Пояснение |
|---|---|---|
| | Терминатор | Начало или конец алгоритма |
| | Процесс | Действие или последовательность действий, обработка данных любого вида |
| | Решение | Блок проверки условия, имеющий один вход и несколько альтернативных выходов, только один из которых может быть активирован после выполнения условия |
| | Предопределенный процесс | Процесс, состоящий из одной или нескольких операций, которые определены в другом месте (в подпрограмме, в модуле) |
| | Ввод или вывод данных | |
| | Начало цикла | Условие для инициализации, приращения, завершения помещаются внутри верхней или нижней фигур в зависимости от расположения операции, проверяющей условие |
| | Конец цикла | |
| | Подготовка | Модификация команды или группы команд с целью воздействия на некоторую последующую функцию |
| | Соединитель | Показывает выход в другую часть алгоритма или вход из другой части алгоритма. Обозначает обрыв линии и продолжение ее в другом месте |
|  | Комментарий | Описательные комментарии или пояснительные записи в целях объяснения или примечания |

Рис. 16. Блок-схемы алгоритмов и программ (по ГОСТ 19.701—90)

языка UML, в частности, диаграмма деятельности (*activity diagram*), диаграмма состояний (*UML state machine diagram*) и некоторые диаграммы взаимодействия (*interaction diagrams*), например, диаграмма синхронизации (*timing diagram*).

## §7. ВЫВОДЫ

1. Знания, выраженные с помощью любого письменного языка, можно разбить на две части:
   - процедурные знания;
   - декларативные знания.
2. Правильно записанные процедурные знания всегда можно превратить в алгоритм.
3. Учащиеся должны уметь:
   - писать, читать и понимать алгоритмы;
   - распознавать алгоритмы, которые встречаются в окружающей жизни, и записывать их.
4. Блок-схемы алгоритмов безнадежно устарели. Пользоваться ими недопустимо. Вместо них следует использовать дракон-схемы.