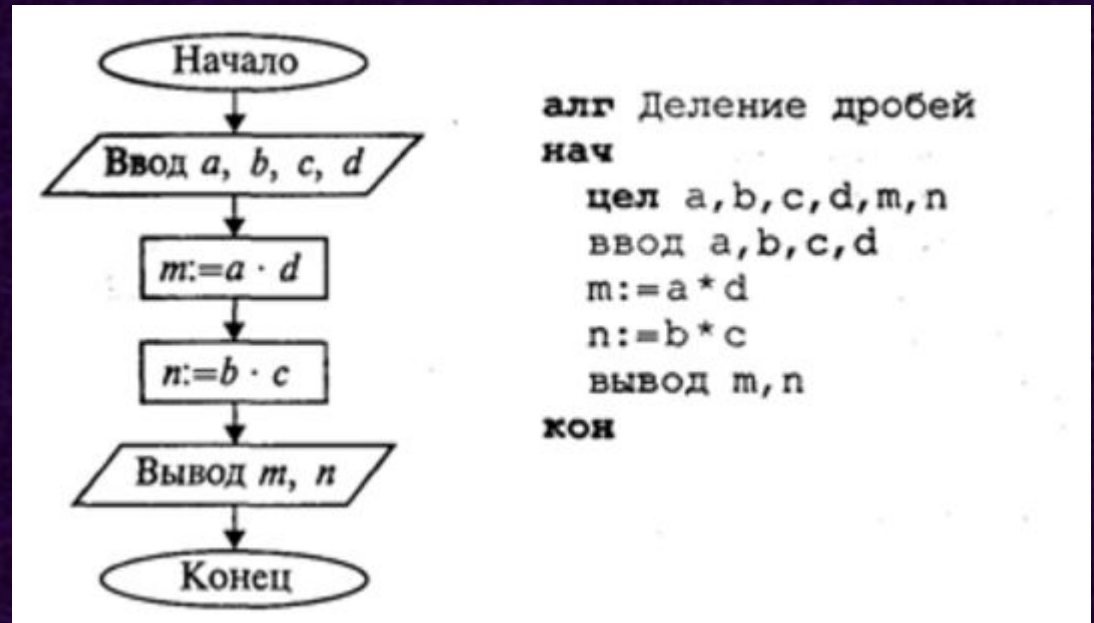
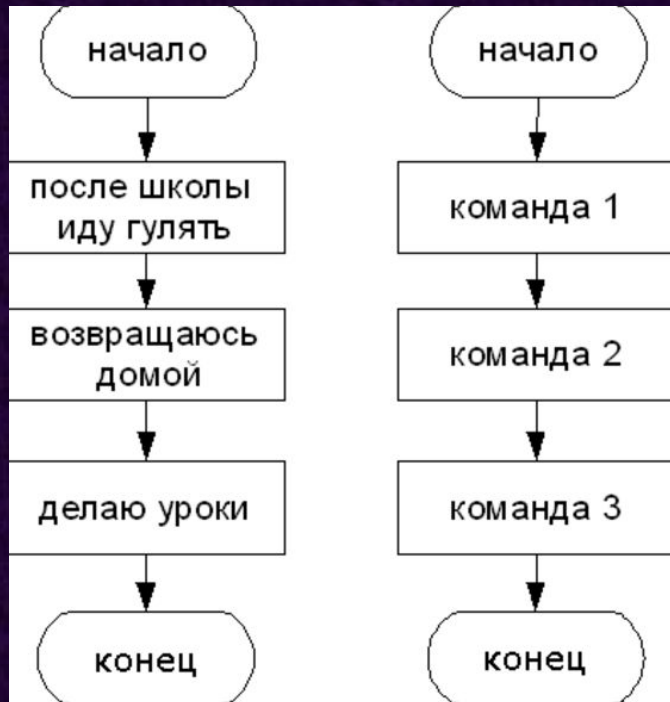


Алгоритмы, структуры алгоритмов, структурное программирование

Понятие об алгоритме:

Алгоритм - это предписание исполнителю (человеку или автомату) выполнить точно определенную последовательность действий, направленных на достижение заданной цели.



Алгоритм - это сформулированное на некотором языке правило, указывающее на действия, последовательное выполнение которых приводит от исходных данных к искомому результату. Значение слова алгоритм очень схоже со значением слов рецепт, процесс, метод, способ. Однако любой алгоритм, в отличие от рецепта или способа, обязательно обладает следующими свойствами.

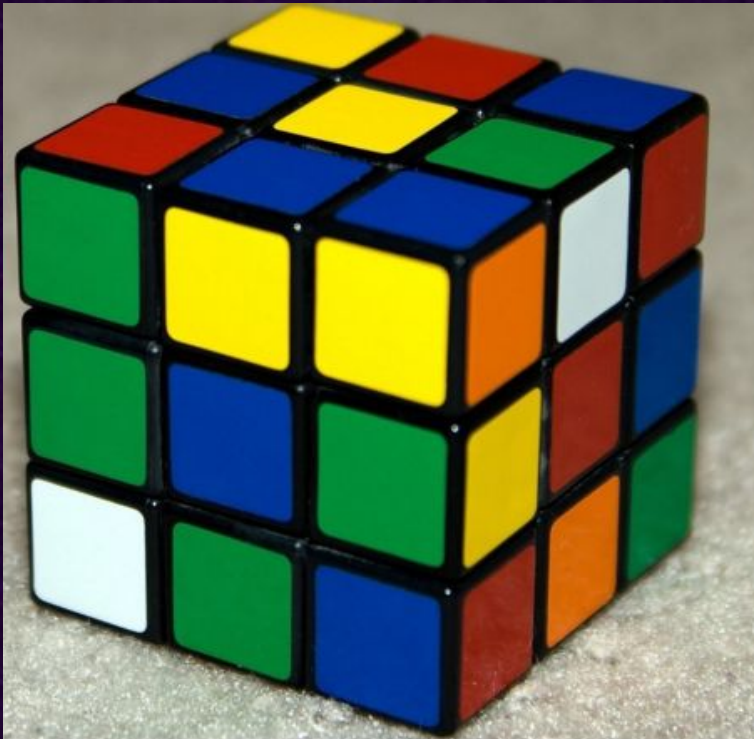
Свойства алгоритма



Основные свойства алгоритма

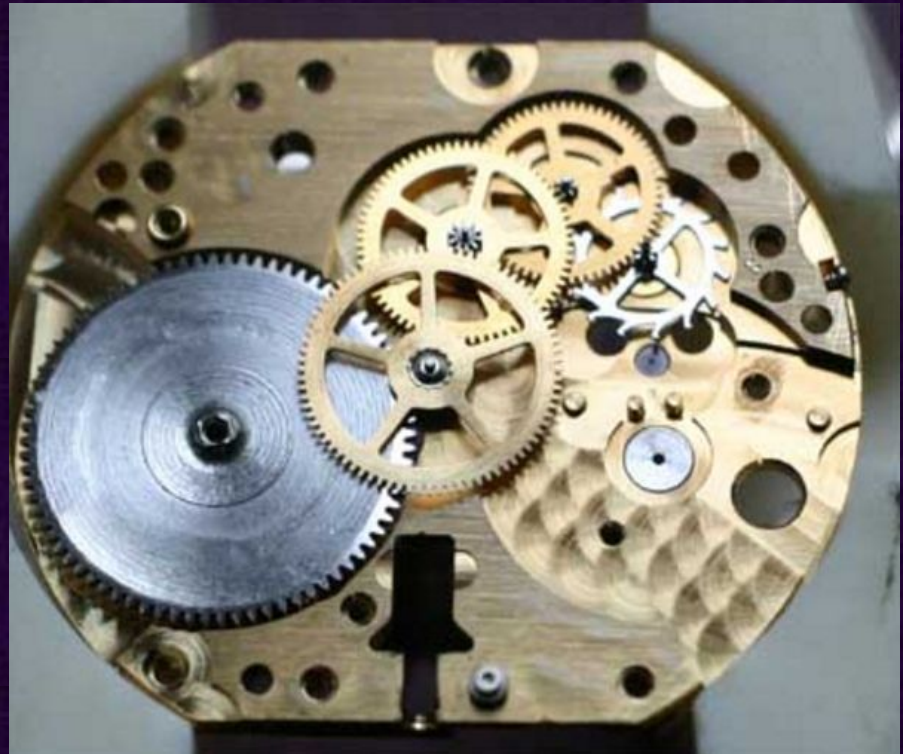
1. Дискретность

- разбиение алгоритма на ряд отдельных законченных действий - шагов. Выполнение алгоритма разбивается на последовательность законченных действий - шагов. Каждое действие должно быть закончено исполнителем алгоритма прежде, чем он приступит к исполнению следующего действия.



2. Точность

- однозначные указания. На каждом шаге однозначно определено преобразование объектов среды исполнителя, полученной на предыдущих шагах алгоритма. Если алгоритм многократно применяется к одному и тому же набору исходных данных, то на выходе он получает каждый раз один и тот же результат. Запись алгоритма должна быть такой, чтобы на каждом шаге его выполнения было известно, какую команду надо выполнять следующей.



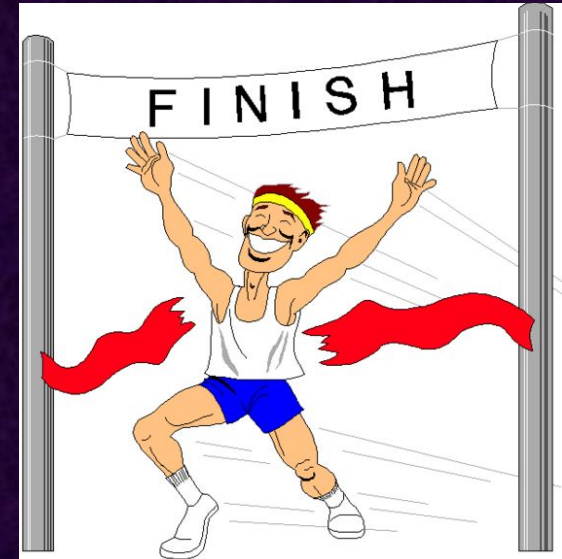
3. Понятность

- однозначное понимание и исполнение каждого шага алгоритма его исполнителем. Алгоритм должен быть записан на понятном для исполнителя языке.



4. Результативность

- обязательное получение результата за конечное число шагов. Каждый шаг (и алгоритм в целом) после своего завершения дает среду, в которой все объекты однозначно определены. Если это по каким-либо причинам невозможно, то алгоритм должен сообщать, что решение задачи не существует. Работа алгоритма должна быть завершена за конечное число шагов. Информатика оперирует только с конечными объектами и конечными процессами, поэтому вопрос о рассмотрении бесконечных алгоритмов остается за рамками теории алгоритмов.



5. Массовость

- применение алгоритма к решению целого класса однотипных задач. Исходные данные могут отличаться.



Исполнитель алгоритма

Исполнитель - это некоторый объект (человек, животное, техническое устройство), способный выполнять определённый набор команд.



Данные и величины

Величины – различные информационные объекты (числа, символы, коды и пр.), с которыми работает универсальный исполнитель алгоритмов – компьютер.

константы

переменные

Данные – совокупность величин.

исходные

промежуточные

результаты

Этапы решения задачи на компьютере

Работа по решению любой задачи с использованием компьютера делится на следующие этапы:

1. Постановка задачи.
2. Формализация задачи.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Часто эту последовательность называют **технологической цепочкой** решения задачи на компьютере.

Разработка алгоритма



Алгоритм – модель деятельности исполнителя алгоритмов

Независимо от того , на каком языке программирования будет написана программа , алгоритм решения любой задачи на компьютере может быть составлен из команд:

- Присваивания;

- Ввода;

- Вывода;

- Обращения в вспомогательному алгоритму (подпрограмме);


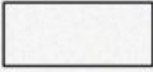

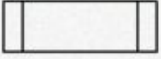
- Цикла;

- Ветвления.

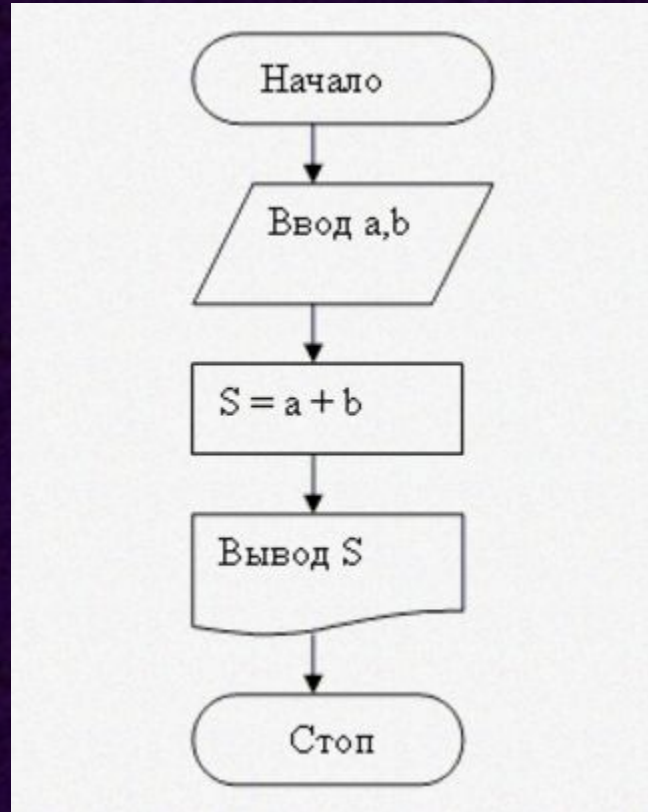
Алгоритмы и величины.

Способы описания алгоритмов.

Выше отмечалось, что один и тот же алгоритм может быть записан по-разному. Можно записывать алгоритм **естественным языком**. В таком виде мы используем рецепты, инструкции и т.п. Для записи алгоритмов, предназначенных формальным исполнителям, разработаны специальные **языки программирования**. Любой алгоритм можно описать **графически в виде блок-схемы**. Для этого разработана специальная система обозначений:

Обозначение	Описание	Примечания
	Начало и конец алгоритма	
	Ввод и вывод данных.	Вывод данных иногда обозначают иначе: 
	Действие	В вычислительных алгоритмах так обозначают присваивание
	Развилка	Развилка - компонент, необходимый для реализации ветвлений и циклов
	Начало цикла с параметром	
	Типовой процесс	В программировании - процедуры или подпрограммы
	Переходы между блоками	

Пример описания алгоритма суммирования двух величин в виде блок-схемы:



Такой способ описания алгоритма наиболее нагляден и понятен человеку. Поэтому, алгоритмы формальных исполнителей обычно разрабатывают сначала в виде блок-схемы, и только затем создают программу на одном из языков программирования.

Типовые алгоритмические структуры.

Программист имеет возможность конструировать и использовать нетипичные алгоритмические структуры, однако, в этом нет необходимости. Любой сколь угодно сложный алгоритм может быть разработан на основе трёх типовых структур: следования, ветвления и повторения. При этом структуры могут располагаться последовательно друг за другом или вкладываться друг в друга.

```
interactive.py
1  """ coding: utf-8 """
2
3  # Copyright (C) 2006-2007 Alec Thomas <alec@svetupoff.org>
4
5  # This software is licensed as described in the file COPYING, which
6  # you should have received as part of this distribution.
7
8
9  """CLY and readline, together at last.
10
11 This module uses readline's line editing and tab completion along w
12 grammar parser to provide an interactive command line environment.
13
14 It includes support for application specific history files, dynamic
15 customisable completion key, interactive help and more.
16
17 Press '?' at any location to contextual help.
18 """
19
20 import os
21 import sys
22 import readline
23 import cly.rtext
24 import cly.console as console
25 from cly.exceptions import Error, ParseError
26 from cly.builder import Grammar
27 from cly.parser import Parser
28
29
30 __all__ = ['Interact', 'interact']
31 __docformat__ = 'restructuredtext en'
32
33
34 class Interact(object):
35     """CLY interaction through readline. Due to readline limitation
36     Interact object can be active within an Application.
37
38     Constructor arguments:
39
40     ``parser``: ``Parser`` or ``Grammar`` object
41         The parser/grammar to use for interaction.
42
43     ``application``: ``cly``: string
44         The application name. Used to construct the history file nam
45     prompt, if not provided.
46
47     ``prompt``: ``None``: string
48         The prompt.
49
50     """
51
52     def __init__(self, grammar_or_parser, application="cly", prompt
53                 user_context=None, with_context=None, history_file
54                 history_length=500, completion_key='tab',
55                 completion_delimiters='\t',
56                 help_key='?', inhibit_exceptions=False,
57                 with_backtrace=False):
58
59         if prompt is None:
60             prompt = application + '> '
61         if history_file is None:
62             history_file = os.path.expanduser('~/.%s_history' % app
63         if isinstance(grammar_or_parser, Grammar):
64             parser = Parser(grammar_or_parser)
65         else:
66             parser = grammar_or_parser
67
68         if with_context is not None:
69             parser.with_context = with_context
70         if user_context is not None:
71             parser.user_context = user_context
72         Interact.parser = parser
73         Interact.prompt = prompt
74         Interact.application = application
75         Interact.user_context = user_context
76         Interact.history_file = history_file
77         Interact.history_length = history_length
78         Interact.completion_delimiters = completion_delimiters
79         Interact.completion_key = completion_key
80
81     def __call__(self):
82
83         readline.set_history_length(history_length)
84         readline.read_history_file(history_file)
85
86         except:
87             pass
88
89         readline.parse_and_bind("?: complete" % completion_key)
90         readline.set_completer_delims(self.completion_delimiters)
91
92
93 """ help_key='?': 'key'
94     Key to use for tab completion.
95 """
96
97 """
98
99 _cli_inject_text = ''
100 _completion_candidates = []
101 _parser = None
102 _prompt = None
103 _user_context = None
104 _history_file = None
105 _application = None
106
107 def __init__(self, grammar_or_parser, application="cly", prompt
108             user_context=None, with_context=None, history_file
109             history_length=500, completion_key='tab',
110             completion_delimiters='\t',
111             help_key='?', inhibit_exceptions=False,
112             with_backtrace=False):
113
114     if prompt is None:
115         prompt = application + '> '
116     if history_file is None:
117         history_file = os.path.expanduser('~/.%s_history' % app
118     if isinstance(grammar_or_parser, Grammar):
119         parser = Parser(grammar_or_parser)
120     else:
121         parser = grammar_or_parser
122
123     if with_context is not None:
124         parser.with_context = with_context
125     if user_context is not None:
126         parser.user_context = user_context
127     Interact.parser = parser
128     Interact.prompt = prompt
129     Interact.application = application
130     Interact.user_context = user_context
131     Interact.history_file = history_file
132     Interact.history_length = history_length
133     Interact.completion_delimiters = completion_delimiters
134     Interact.completion_key = completion_key
135
136     def __call__(self):
137
138         readline.set_history_length(history_length)
139         readline.read_history_file(history_file)
140
141         except:
142             pass
143
144         readline.parse_and_bind("?: complete" % completion_key)
145         readline.set_completer_delims(self.completion_delimiters)
146
147 """
```

Следование

Следование - алгоритмическая конструкция, отображающая естественный, последовательный порядок действий.

Алгоритмы, в которых используется только структура «следование», называются *линейными*.

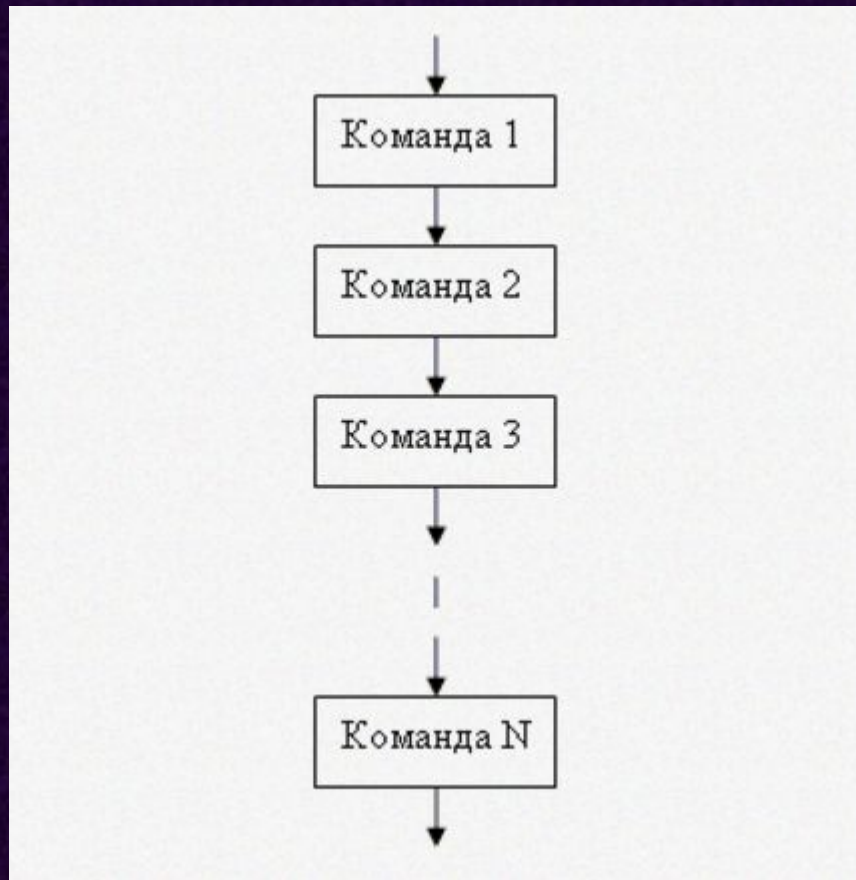
```
graph TD; A[Действие 1] --> B[Действие 2];
```

Действие 1

Действие 2

Линейная структура (следование)

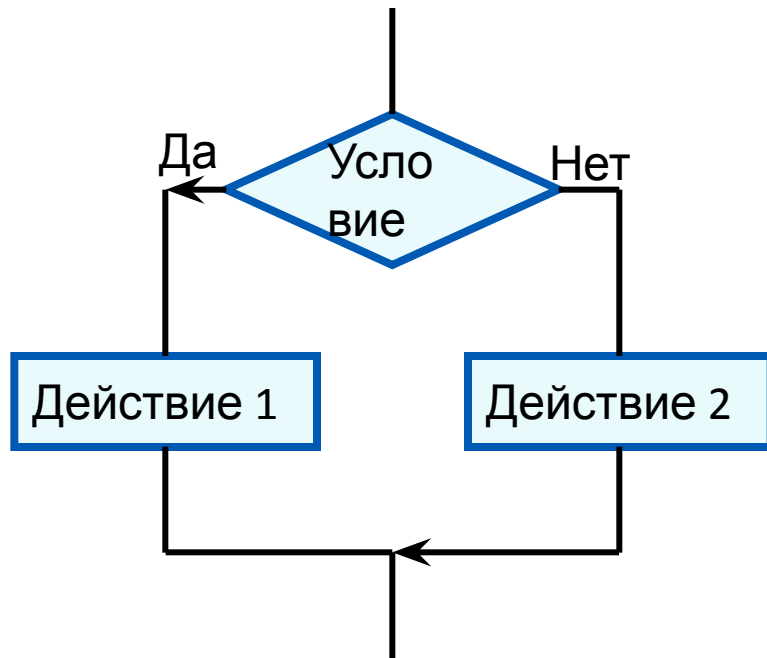
Наиболее простой алгоритмической структурой является **линейная**. В ней все операции выполняются один раз в том порядке, в котором они записаны.



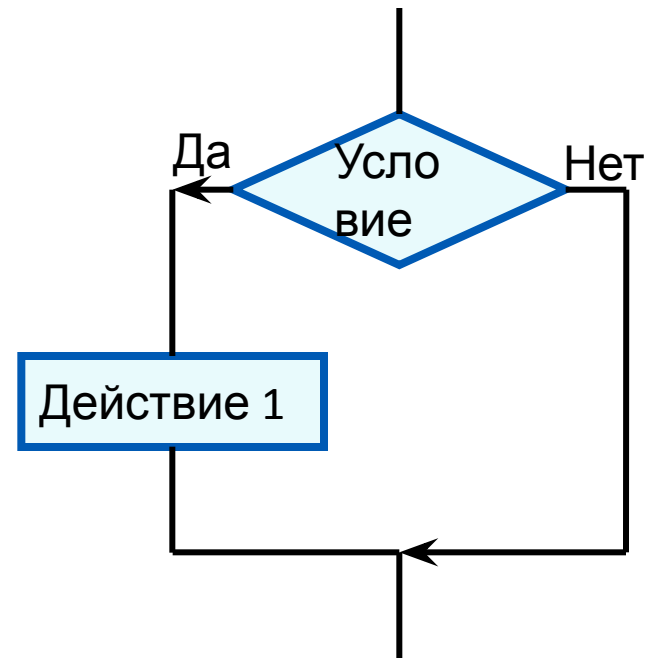
Ветвление

Ветвление - алгоритмическая конструкция, в которой в зависимости от результата проверки условия (да или нет) предусмотрен выбор одной из двух последовательностей действий (ветвей).

Алгоритмы, в основе которых лежит структура «ветвление», называются **разветвляющимися**.



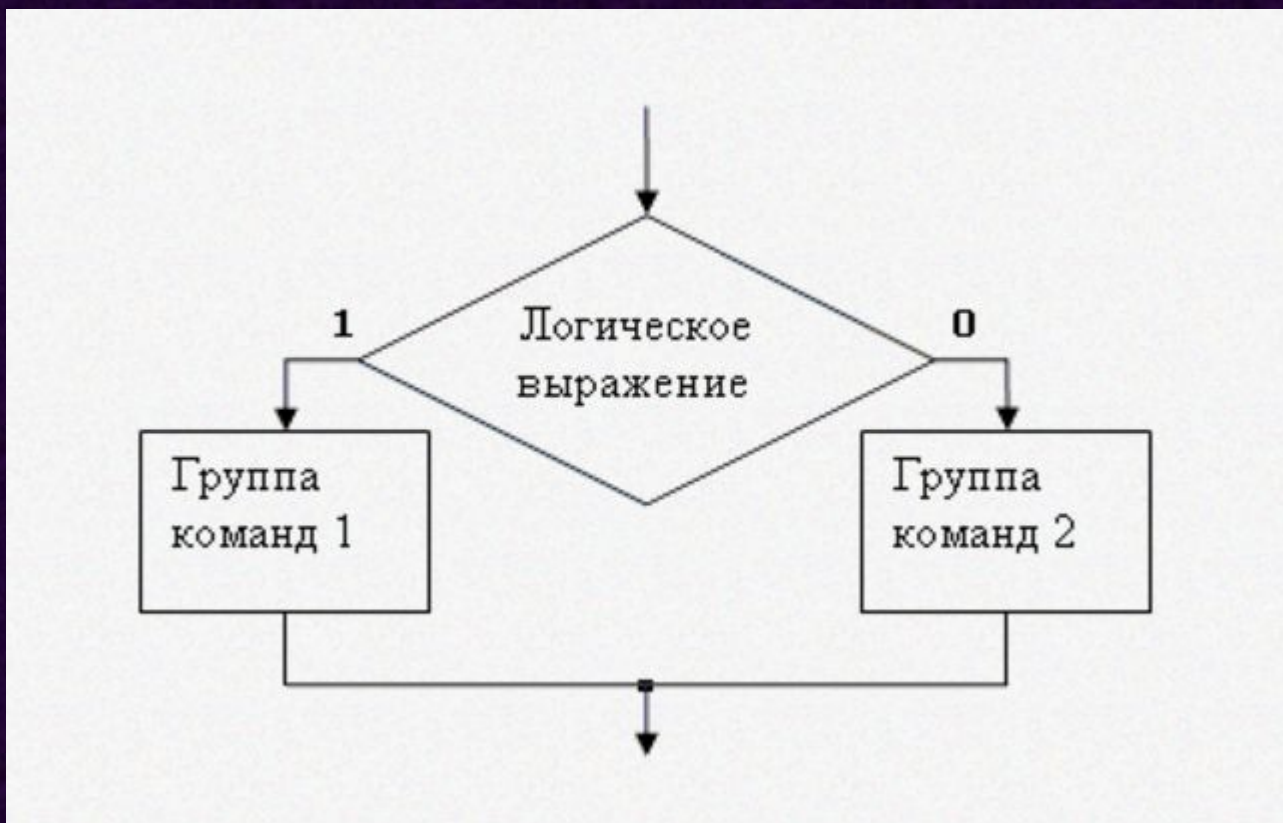
Полная форма ветвления



Неполная форма ветвления

Ветвление

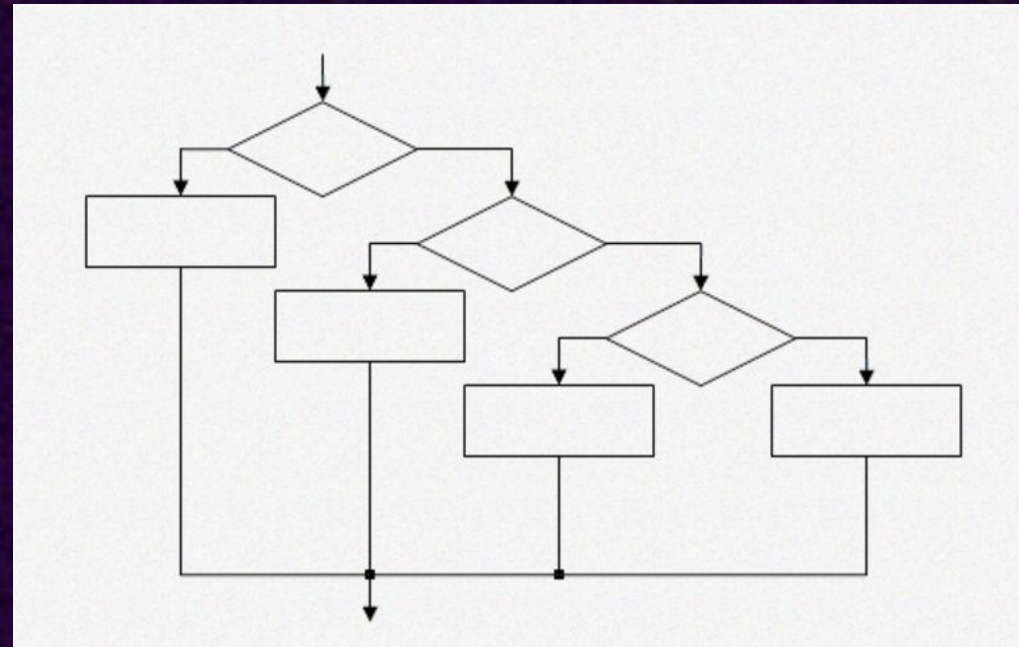
В полном ветвлении предусмотрено два варианта действий исполнителя в зависимости от значения логического выражения (условия). Если условие истинно, то выполняться будет только первая ветвь, иначе только вторая ветвь.



Вторая ветвь может быть пустой. Такая структура называется **неполным ветвлением** или **обходом**.



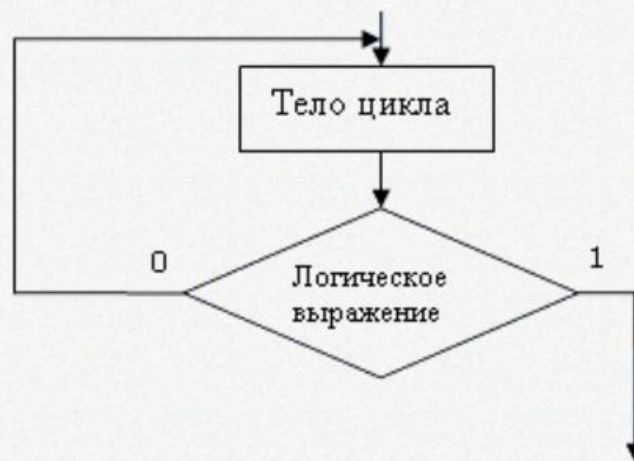
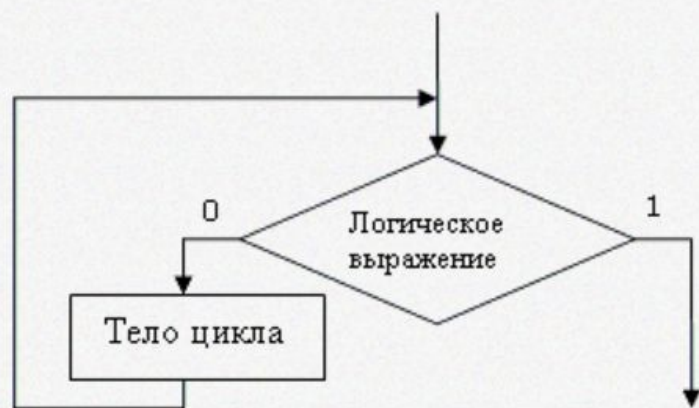
Из нескольких ветвлений можно сконструировать структуру «**выбор**» (множественное ветвление), которая будет выбирать не из двух, а из большего количества вариантов действий исполнителя, зависящих от нескольких условий. Существенно, что выполняется только одна ветвь - в такой структуре важное значение приобретает порядок следования условий: если выполняются несколько условий, то сработает только одно из них - первое сверху.



Цикл (повторение)

Цикл позволяет организовать многократное повторение одной и той же последовательности команд - она называется телом цикла. В различных видах циклических алгоритмов количество повторений может зависеть от значения логического выражения (условия) или может быть жестко задано в самой структуре. Различают циклы : «до», «пока», циклы со счётчиком. В циклах «до» и «пока» логическое выражение (условие) может предшествовать телу цикла (цикл с предусловием) или завершать цикл (цикл с послеусловием).

Циклы «до» - повторение тела цикла до выполнения условия:



Цикл (повторение)

Повторение - алгоритмическая конструкция, представляющая собой последовательность действий, выполняемых многократно.

Алгоритмы, содержащие конструкцию «повторение», называются **циклическими** или **циклами**.

Последовательность действий, многократно повторяющаяся в процессе выполнения цикла, называется **телом цикла**.

