Лекция 1. Алгоритм. Свойства алгоритмов. Блок-схема. Нисходящее и восходящее проектирование. Абстракция.

Вид занятия	Частота	Вид аттестации	
		I семестр	II семестр
Лекция	1 раз/1 нед	Экзамен	Экзамен
Семинар	1 раз/2 нед	КР1 КР2	
Лабораторные	1 раз/1 нед	Допуск	Допуск

I семестр – Алгоритмизация, программирование на языке *C* в консольном приложении. Структуры данных: простые и статические массивы, текстовые файлы. Алгоритмы: вычисления суммы, произведения, количества, поиска значения и номера экстремального значения, сортировки, проверки условий, выполняемых «для всех»/ «хотя бы одного» элемента массива.
II семестр – Алгоритмизация и процедурное программирование на языке *C|C++*. Структуры данных: структуры, динамические массивы, списки, стеки, деревья, строки, файлы. Алгоритмы: поиска корня уравнения на отрезке с заданной точностью и суммы бесконечного ряда, рекурсия.

Алгоритм. Свойства алгоритмов

Алгоритм – строго определенная последовательность действий, формальное выполнение которой приводит к решению поставленной задачи за конечное число шагов.
Алгоритмизация – процесс разработки алгоритма (плана действий) для решения задачи.

Появление алгоритмов связывают с зарождением математики. Более 1000 лет назад (в 825 году) ученый из города Хорезма Абдулла (или Абу Джафар) Мухаммед бен Муса аль-Хорезми создал книгу по математике, в которой описал *способы выполнения арифметических действий над многозначными числами*. Само слово «алгоритм» возникло в Европе после перевода на латынь книги этого среднеазиатского математика, в которой его имя писалось как «Алгоритми».

Человеку в жизни и практической деятельности приходится решать множество различных задач. Решение каждой из них описывается своим алгоритмом, разнообразие этих алгоритмов очень велико. Можно выделить три основных вида алгоритмов:

1. **линейной структуры,**
2. **разветвляющейся структуры,**
3. **циклической структуры**.

Для краткости их называют просто: линейные, разветвляющиеся и циклические алгоритмы. Разнообразие же алгоритмов определяется тем, что любой алгоритм распадается на части, фрагменты и каждый фрагмент представляет собой алгоритм одного из трех указанных видов. Поэтому важно знать структуру каждого из алгоритмов.

В алгоритмах *линейной* структуры действия выполняются последовательно одно за другим:

В алгоритмах *разветвленной* структуры в зависимости от выполнения или невыполнения какого-либо условия производятся различные последовательности действий. Каждая такая последовательность действий называется *ветвью алгоритма*.

В алгоритмах *циклической* структуры в зависимости от выполнения или невыполнения какого-либо условия выполняется повторяющаяся последовательность действий, называющаяся телом цикла. *Вложенным* называется цикл, находящийся внутри тела другого цикла.

Различают циклы с *предусловием* и *постусловием*:

Итерационным называется цикл, *число повторений которого не задается*, а определяется в ходе выполнения цикла. В этом случае одно повторение цикла называется *итерацией*.

4. **Вспомогательный алгоритм –** алгоритм, который можно использовать в других алгоритмах, указав только его имя. Например: вы в детстве учились суммировать единицы, затем десятки, чтобы суммировать двузначные числа содержащие единицы вы не учились новому методу суммирования, а воспользовались старыми методами.

Свойства алгоритмов

Понятность – каждая команда должна входить в систему команд исполнителя.

Дискретность – это разбиение алгоритма на ряд отдельных законченных команд **(шагов**), каждая из которых должна быть выполнена прежде, чем исполнитель перейдет к выполнению следующей.

Детерминированность (точность, определенность) – команда алгоритма исполнителем должна пониматься однозначно. Не должно быть двоякого толкования команды.

Результативность и **конечность** – за конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.

Массовость – алгоритм решения задачи разрабатывается в общем виде – он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. (Исходные данные могут выбираться из некоторой области, которая называется *областью применимости алгоритма*.)

Рассмотрим свойства алгоритмов на примерах:

Пример. алгоритм выполнения открывания двери.
1. Достать ключ из кармана.
2. Вставить ключ в замочную скважину.
3. Повернуть ключ два раза против часовой стрелки.
4. Вынуть ключ.

Вас пригласили в гости и подробно объяснили, как добраться:
1. Выйти из дома.
2. Повернуть направо.
3. Пройти два квартала до остановки.
4. Сесть в автобус № 5, идущий к центру города.
5. Проехать три остановки.
6. Выйти из автобуса.
7. Найти по указанному адресу дом и квартиру.

Понятность – каждая команда должна входить в систему команд исполнителя.

***Дискретность (от лат. discretus — разделенный, прерывистый) – это разбиение алгоритма на ряд отдельных законченных действий (шагов).* В приведенных выше алгоритмах общим является необходимость строгого соблюдения последовательности выполнения действий. Попробуем *переставить* в первом примере второе и третье действия. Вы, конечно, сможете выполнить и этот алгоритм, но дверь вряд ли откроется. А если поменять местами, предположим, пятое и второе действия во втором примере, алгоритм станет невыполнимым.**

***Детерминированность (от лат. determinate — определенность, точность) – любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае.* Например, если к остановке подходят автобусы разных маршрутов, то в алгоритме должен быть указан *конкретный номер* маршрута — 5. Кроме того, необходимо указать *точное количество* остановок, которое надо проехать, — скажем, три.**

***Конечность – каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения.* В приведенных примерах каждое описанное действие реально и может быть выполнено. Поэтому и алгоритм имеет предел, то есть конечен.**

Результативность*. В алгоритме должны быть предусмотрены все возможные пути решения, в том числе *альтернативные*. Пример: алгоритм нахождения *большего из двух заданных чисел A и B:
1. Из числа A вычесть число B.
2. Если получилось отрицательное значение, то сообщить, что число B больше.
3. Если получилось положительное значение, то сообщить, что число A больше.

При всей простоте и очевидности алгоритма, не каждый сразу поймет его *ошибочность*. Ведь если оба числа равны, то не получится *никакого сообщения*. Значит, надо обязательно предусмотреть это вариант, например:
1. Из числа A вычесть число B.
2. Если получилось отрицательное значение, то сообщить, что число B больше.
3. Если получилось положительное значение, то сообщить, что число A больше.
4. Если получился ноль, то сообщить, что числа равны.

Массовость – один и тот же алгоритм можно использовать с разными исходными данными.
Например: *алгоритм приготовления любого бутерброда.*
1. Отрезать ломтик хлеба.
2. Намазать его маслом.
3. Отрезать кусок любого другого пищевого продукта (колбасы, сыра, мяса).
4. Наложить отрезанный кусок на ломоть хлеба.

Возможность автоматизации деятельности человека.

Выполняя алгоритм, *исполнитель может не вникать в смысл того, что он делает, и вместе с тем получать нужный результат*. В таком случае говорят, что исполнитель действует ***формально*** (механически), т. е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.

Построение алгоритма для решения задачи из какой-либо области *требует от человека глубоких знаний* в этой области, бывает связано с *тщательным анализом* поставленной задачи, сложными, иногда очень громоздкими *рассуждениями*. На поиски алгоритма решения некоторых задач ученые затрачивают многие годы. ***Но когда алгоритм создан, решение задачи по готовому алгоритму уже не требует каких-либо рассуждений и сводится только к строгому выполнению команд алгоритма***.

В этом случае исполнение алгоритма можно **поручить не человеку, а машине**. Действительно, простейшие операции, на которые при создании алгоритма расчленяется процесс решения задачи, может реализовать и машина, специально созданная для выполнения отдельных команд алгоритма и выполняющая их в последовательности, указанной в алгоритме. Это положение и лежит в основе работы автоматических устройств, автоматизации деятельности человека.

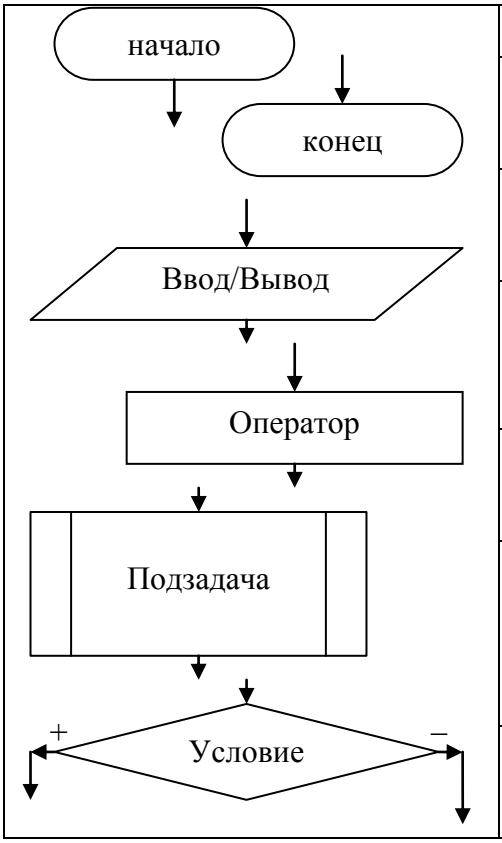
Способы описания алгоритмов

Словесный (для записи используются специальные формальные языки с ограниченным набором слов и строгими правилами записи. Представляет собой описание последовательных этапов обработки данных), **Формульный**, **Словесно-формульный**, **Графический** (изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий – ***Блок-схемы***).

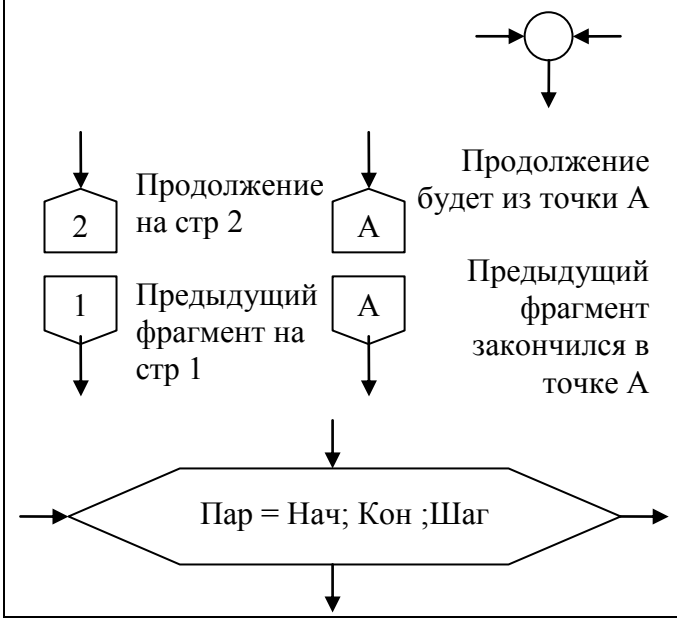
Блок-схемы позволяют описать алгоритм с использованием минимального *количества слов и независимо от выбранного процедурного языка*. Порядок действий указывается с помощью стрелок.

Алгоритм. Блок-схема

Основные блоки

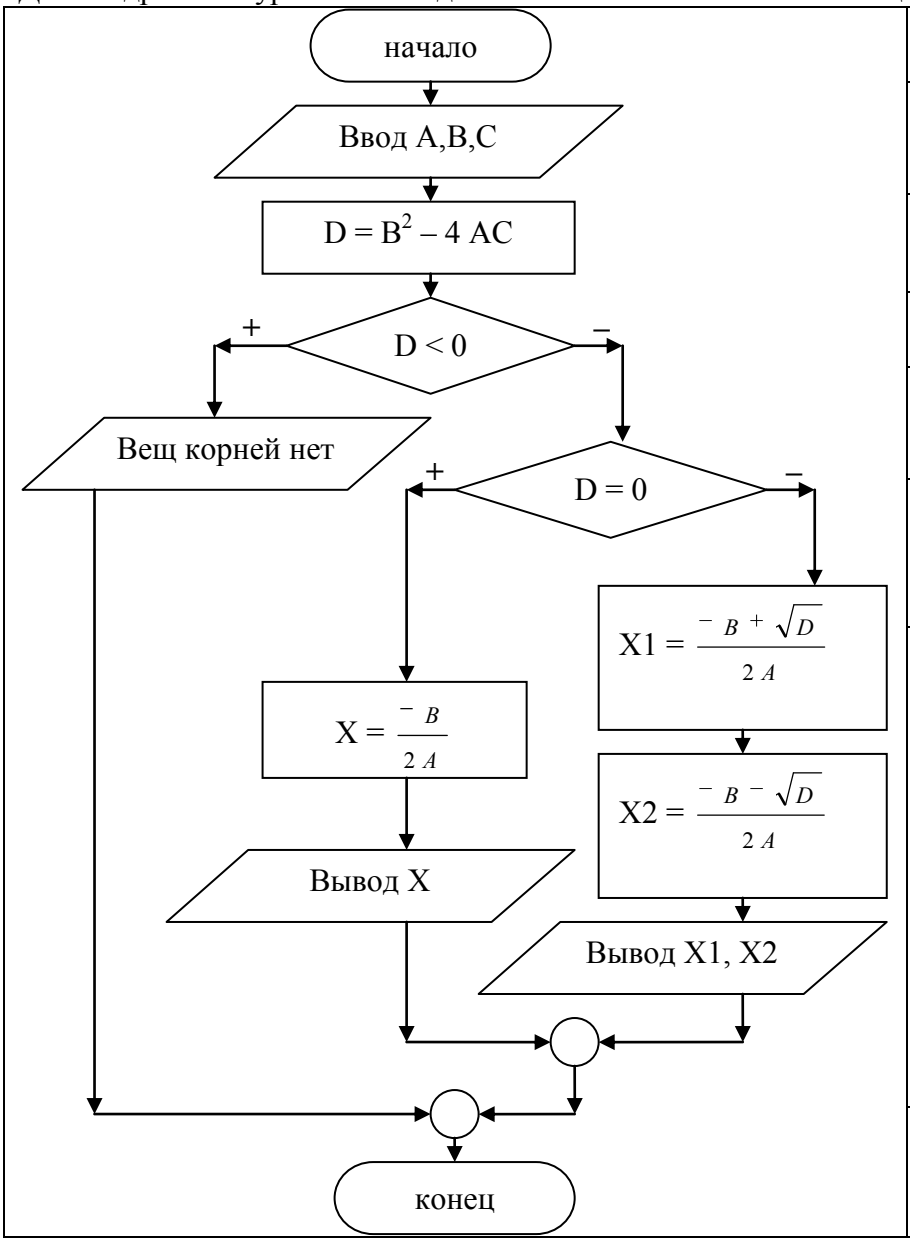
	Описание
	Начало – с этого блока начинается алгоритм НЕТ входов, ОДИН выход
	Конец – этим блоком заканчивается алгоритм ОДИН вход, НЕТ выходов
	Ввод/Вывод – блок ввода/вывода значений переменных без уточнения способа (клавиатура, экран, файл) ОДИН вход, ОДИН выход
	Оператор – блок для одного оператора – чаще всего оператора присваивания ОДИН вход, ОДИН выход
	Подзадача – вспомогательный алгоритм, который описывается отдельной блок-схемой, а в программе часто – отдельной подпрограммой (процедурой или функцией). ОДИН вход, ОДИН выход
	Условие – блок проверки условия. *Условие* может принимать значение *Истина* или *Ложь* ОДИН вход, ДВА выхода, помеченных +/– или Да/Нет

Вспомогательные блоки

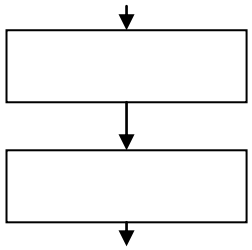
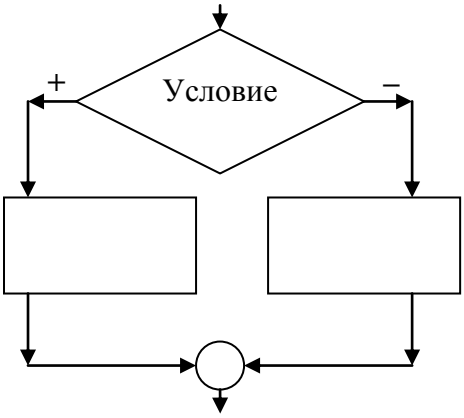
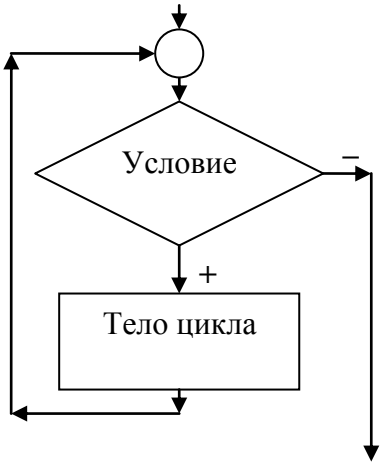
	Узел – объединение ветвей, можно обозначать *точкой* ДВА входа, ОДИН выход
	Ссылка на продолжение алгоритма, если он не уместился на одной странице Указывается либо номер страницы (1, 2, 3…), либо точки перехода (А, В, С… А, Б, В…) ОДИН вход, НЕТ выходов
	Продолжение – ссылка на предыдущую часть блок-схемы НЕТ входа, ОДИН выход
	Блок для обозначения параметрического цикла. Подробности см. ниже ДВА входа и ДВА выхода

Пример. Алгоритм вычисления корней квадратного уравнения

Для квадратного уравнения вида $Ax^2 + Bx + C = 0$ найти все вещественные корни

	Начало алгоритма
	Ввод необходимых данных: A,B,C
	Поиск дискриминанта D (промежуточный результат)
	Проверка отсутствия вещественных корней
	Если условие (D < 0) истинно, то Вывод сообщения об их отсутствии и Конец
	Иначе - Проверка совпадения корней (D=0)
	Две ветви: Если корни совпадают, То ищем и выводим единственное значение корня и Конец Иначе – Ищем и выводим значения обоих корней и Конец
	Конец алгоритма

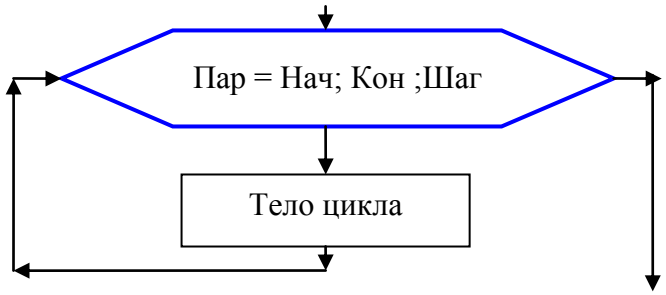
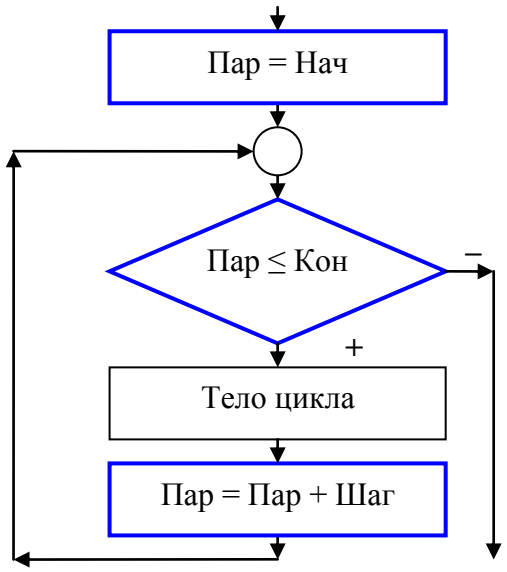
Основные управляющие конструкции

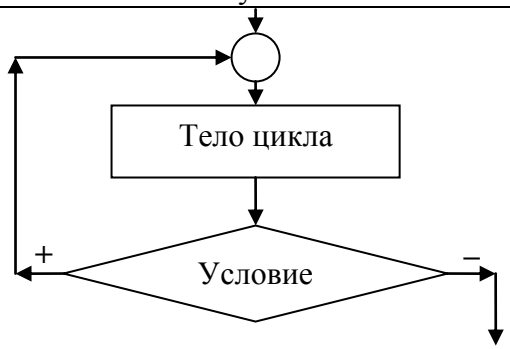
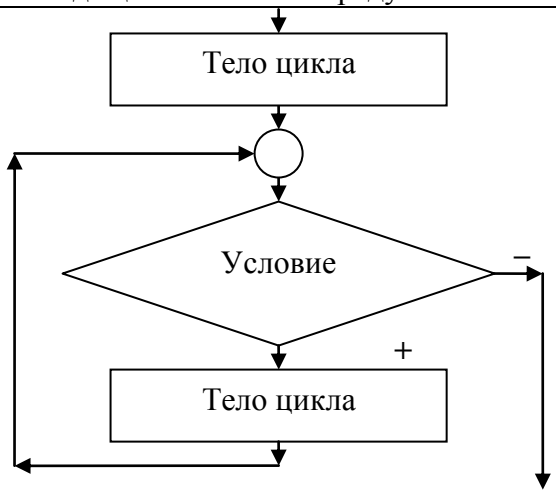
Следование	Ветвление	Повторение (Цикл ПОКА)
		
Два и более подряд идущих блока одного или разного вида выполняются один за другим.	Разветвление алгоритмы в зависимости от текущего значения указанного *Условия* с последующим объединением. На одной из ветвей операторы могут отсутствовать. Операторы по ветви, помеченной знаком +, выполняются только в том случае, если выполняется *Условие*. Операторы по ветви, помеченной знаком –, выполняются только в том случае, если *Условие* НЕ выполняется. После выполнения операторов одной из ветвей выполняются операторы, следующие за данной конструкцией.	Повторение выполнения операторов *Тела цикла*, **ПОКА** выполняется указанное *Условие*. Если *Условие* не выполняется, начинают выполняться операторы, следующие за эти циклом *Тело цикла* может не выполнится ни разу Это цикл с *пред*условием

Принцип структурного программирования:

Этих трех конструкций достаточно для описания любого алгоритма!

Дополнительные виды циклов

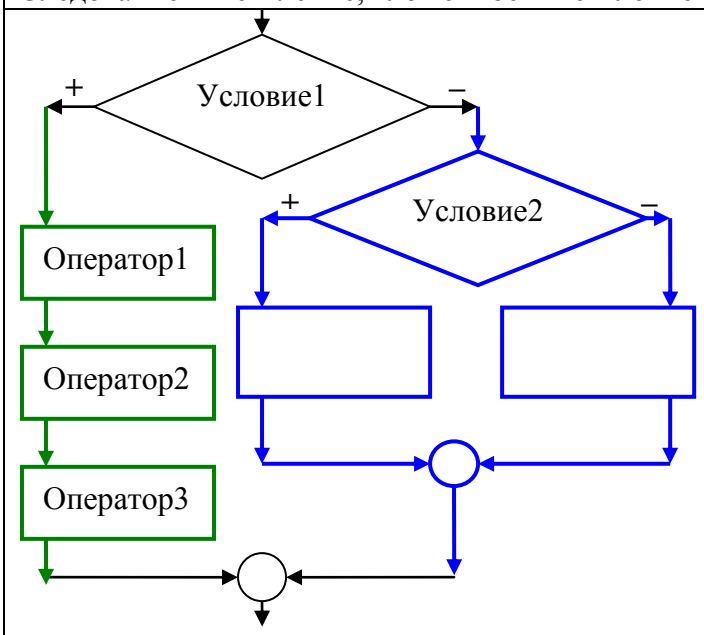
Параметрический (Цикл ДЛЯ)	Полный вид (в виде цикла ПОКА)
 Является краткой (1 новый блок вместо трех) записью для цикла с *пред*условием, управляемым целочисленным параметром, меняющим значение с *Нач* по *Кон* с шагом *Шаг* Например, *тело цикла* с параметром I=1;10;+1 выполнится 10 раз. При первом выполнении I=1, при втором I=2, на 10 шаге I=10. То есть цикл выполнится **ДЛЯ** I = 1,2,3,…10	Для положительного шага:  *Тело цикла* может не выполнится ни разу, если при положительном шаге Нач>Кон

Цикл ПОКА с *пост*условием	В виде цикла ПОКА с предусловием:
 Перед проверкой *Условия* выполняется *Тело цикла*. Выполнение *Тела цикла* осуществляется, пока У*словие* истинно. В *Pascal* в цикле с *постусловием*, *Тело цикла* выполняется **ДО** тех пор, пока Условие не станет истинным. В *Basic* есть обе разновидности. В *Python* нет с постусловием.	 *Тело цикла* выполняется хотя бы 1 раз.

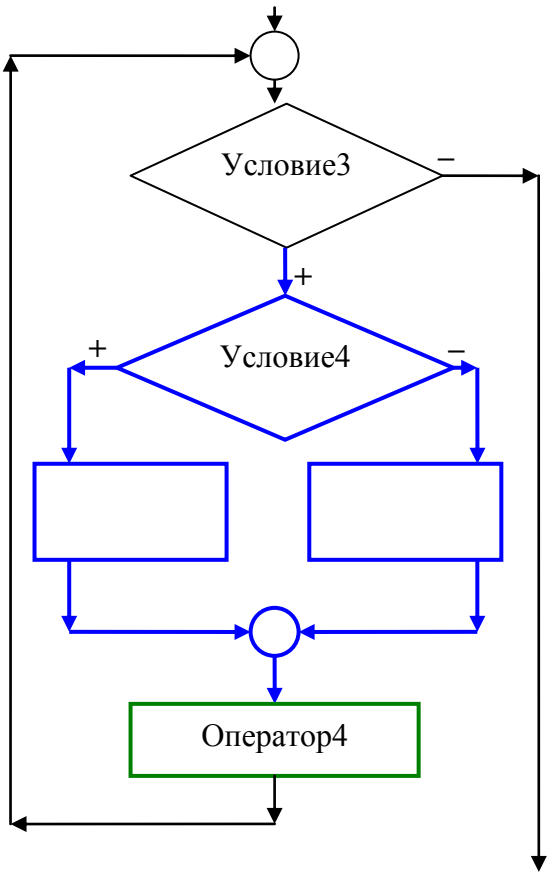
Комбинации конструкций

Конструкции могут **следовать** друг за другом или **вкладываться** друг в друга. Конструкции вкладываются друг в друга ЦЕЛИКОМ!

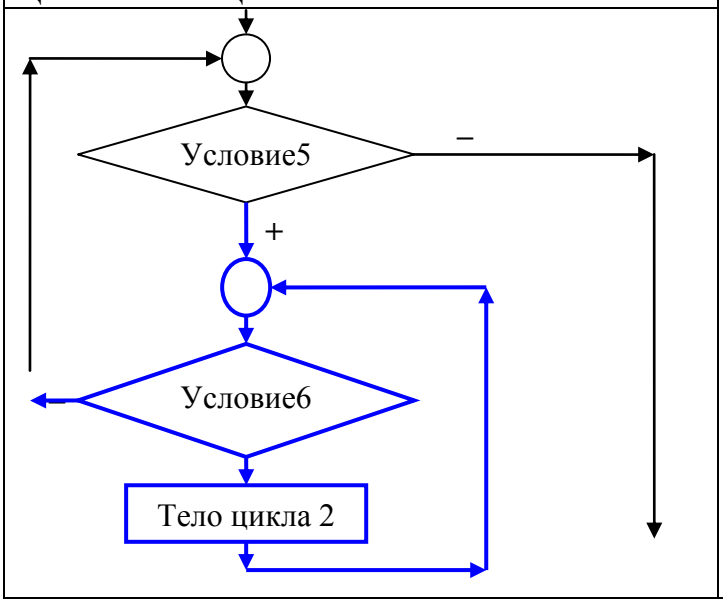
Следование и Ветвление, вложенное в Ветвление



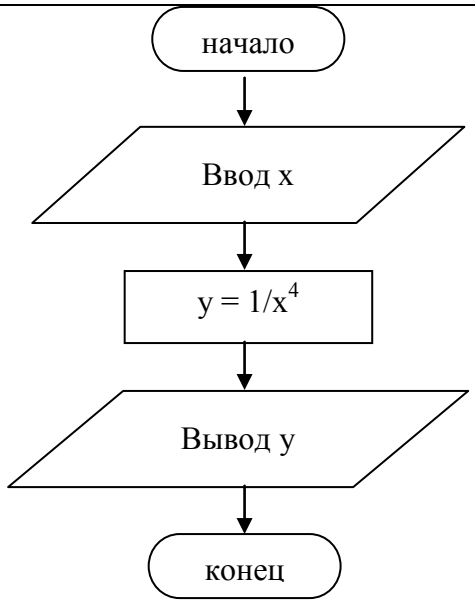
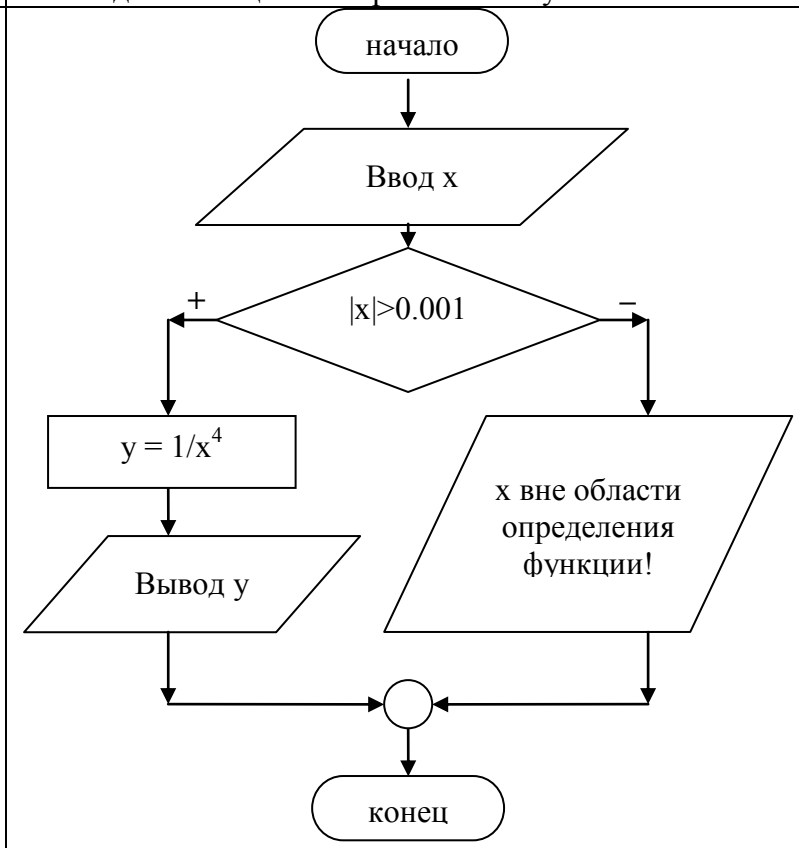
Ветвление и Следование в Цикле ПОКА



Цикл ПОКА в Цикле ПОКА

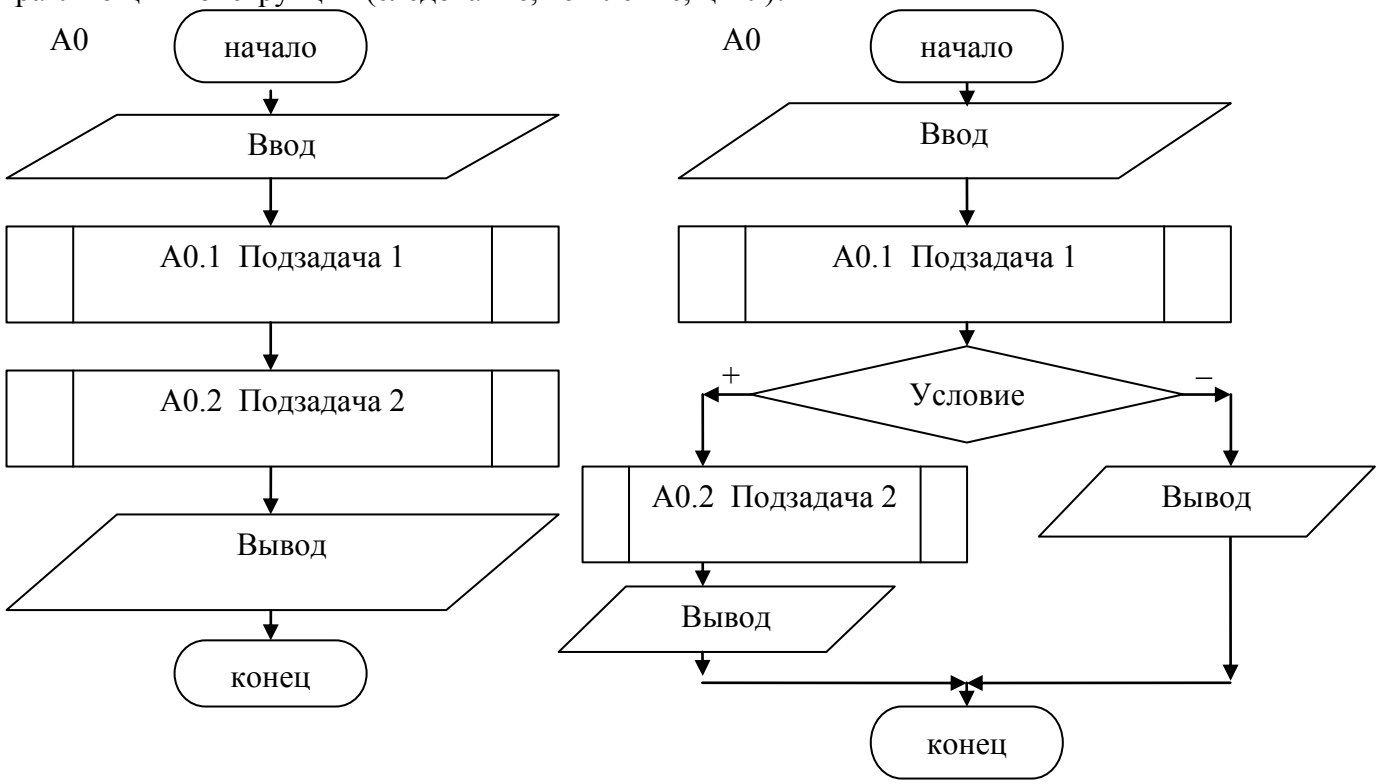


Пример алгоритма (первая лабораторная работа)

Линейный алгоритм	**Ветвящийся**
Вычисление значения функции $y=1/x^4$	Вычисление значения функции $y=1/x^4$, только для x, лежащих в *области определения* функции, и выводом сообщения в противном случае.
	

Нисходящее и восходящее проектирование. Абстракция.

Только простейшие алгоритмы можно написать «с ходу», остальные следует разбивать на подзадачи, **избегая сильной детализации** на первом же шаге. Надо ***абстрагироваться*** от деталей решения задачи и описать ее *решение в общих чертах*, объединяя ***абстракции*** с помощью управляющих конструкций (следование, ветвление, цикл):



Затем, ***постепенно также разбивать полученные задачи(абстракции) на подзадачи, увеличивая детализацию на каждом шаге***. Такой порядок проектирования называется ***нисходящим***. Обратный – от подзадач к основной задаче – ***восходящим***.

