

# «Алгоритмы и методы вычислений»

**Кожемякин Руслан Александрович**

**Цель курса** – создать алгоритмическую основу знаний, которая в дальнейшем понадобится при изучении других учебных курсов и при решении практических задач на будущем месте работы.

# Тема лекции:

***Понятие термина «алгоритм»,  
его виды,  
характеристики,  
вычислительная сложность  
алгоритмов***

# Что такое алгоритмы?

**Алгоритм** - набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата.

**Алгоритм** – это любая корректно определенная вычислительная процедура, на **вход** которой подается величина или набор величин, а результатом выполнения является **выходная** величина или набор значений.

Фактически понятие **алгоритм** можно рассматривать как инструмент для решения поставленной вычислительной задачи.





# Виды алгоритмов

Выделим основные виды алгоритмов которые встречаются на практике:

- 1) **Линейный алгоритм** — набор команд (указаний), выполняемых *последовательно* во времени друг за другом.
- 2) **Разветвляющийся алгоритм** — алгоритм, содержащий хотя бы одно условие, в результате проверки которого может осуществляться *разделение* на несколько альтернативных ветвей алгоритма
- 3) **Циклический алгоритм** — алгоритм, предусматривающий *многократное повторение* одних и тех же операций над данными до удовлетворения некоторого условия.
- 4) **Структурная блок-схема, граф-схема** — графическое изображение алгоритма в *виде схемы* связанных между собой с помощью стрелок (линий перехода) блоков — графических символов, каждый из которых соответствует одному шагу алгоритма.

# Вычислительная сложность алгоритмов

Распространенным критерием оценки алгоритмов является **время работы** и порядок роста продолжительности работы в зависимости от объёма входных данных.

Время, которое тратит алгоритм как функция от размера задачи  $n$  называют временной сложностью этого алгоритма  $T(n)$ . Асимптотику поведения этой функции при увеличении размера задачи называют асимптотической временной сложностью, а для её обозначения используют **специальную нотацию**.

Под **асимптотикой** понимается характер изменения функции при её стремлении к определённой точке.



Обозначение	Граница
(Тета) $\Theta$	Нижняя и верхняя границы, точная оценка
(O - большое) $O$	Верхняя граница, точная оценка неизвестна
(o - малое) $o$	Верхняя граница, не точная оценка
(Омега - большое) $\Omega$	Нижняя граница, точная оценка неизвестна
(Омега - малое) $\omega$	Нижняя граница, не точная оценка

# Вычислительная сложность алгоритмов

Сложность	Комментарий
$O(1)$	Устойчивое время работы не зависит от размера задачи
$O(\log \log n)$	Очень медленный рост необходимого времени
$O(\log n)$	Логарифмический рост — удвоение размера задачи увеличивает время работы на постоянную величину
$O(n)$	Линейный рост — удвоение размера задачи удвоит и необходимое время
$O(n \log n)$	Линеаритмичный рост — удвоение размера задачи увеличит необходимое время чуть более, чем вдвое
$O(n^2)$	Квадратичный рост — удвоение размера задачи увеличивает необходимое время в четыре раза
$O(n^3)$	Кубичный рост — удвоение размера задачи увеличивает необходимое время в восемь раз
$O(c^n)$	Экспоненциальный рост — увеличение размера задачи на 1 приводит к $c$ -кратному увеличению необходимого времени; удвоение размера задачи увеличивает необходимое время в квадрат

# Для чего нужна алгоритмическая база?

Хорошая алгоритмическая база составляет примерно 30% квалификации опытного программиста, в то время как знание синтаксиса и библиотек языка программирования – лишь около 10%.

Решение любой задачи программиста состоит из трех основных этапов:

**1.Классификация.** Необходимо определить к какому из классов алгоритмов относится данная задача. При необходимости задачу можно предварительно разбить на подзадачи и решать каждую из них по отдельности. На этом этапе выясняется, есть ли у задачи эффективное (быстрое) решение или же не существует алгоритма, решающего задачу за приемлемое время.

**2.Формализация и выбор алгоритма.** Выясняется, каких исходных данных не хватает для решения задачи, какие задачи должны быть решены предварительно. Выбирается алгоритм для решения задачи. Обычно выбор идет между алгоритмом, простым для написания кода, но медленным в работе, и алгоритмом, сложным в реализации, но быстрым в работе или эффективным по другим показателям (требования к объему памяти и др.).

**3.Реализация** (программирование) выбранного алгоритма. При необходимости можно почитать (вспомнить), как он работает.

Нет никакой необходимости учить алгоритмы наизусть. Лучше хорошо ориентироваться в алгоритмах, понимать принципы их работы, уметь быстро оценивать реальность решения той или иной задачи и оценивать время, которое потребуется на программирование ее решения.



# Задачи решаемые с помощью алгоритмов

Практическое применение алгоритмов чрезвычайно широко:

- 1) Проект по расшифровке генома человека** далеко продвинулся к своей цели – идентификации всех генов входящих в состав ДНК человека, определению последовательностей, образуемых 3 миллиардами базовых пар, из которых состоит ДНК, к сортировке этой информации в базах данных и разработке инструментов для ее анализа.
- 2) Интернет** – определение оптимальных маршрутов, по которым перемещаются данные, быстрый поиск страниц.
- 3) Электронная коммерция** – существенно зависит от способности защищать информацию (номера кредитных карт, паролей, счетов). В число таких алгоритмов входят криптография, цифровые подписи и др.
- 4) Навигация** (google-maps, яндекс-карты и др. сервисы) – как найти наиболее короткий путь между двумя точками с учетом местности, дорог и др.

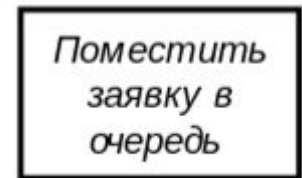




# Блок-схемы алгоритмов

**Блок-схема** — распространенный тип *схем* (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности.

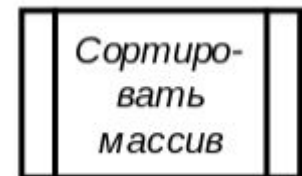
**Элемент «процесс».** Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).



**Элемент «данные».** Символ отображает данные, носитель данных не определен. Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет носителя данных (для указания типа носителя данных используются специфические символы).

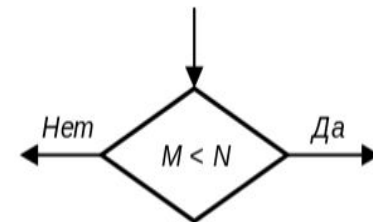


**Элемент «процесс».** Символ отображает predetermined process, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле). Например, вызов процедуры или функции.

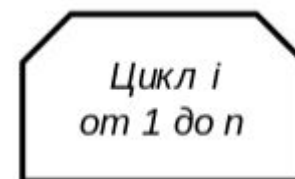


# Блок-схемы алгоритмов

**Элемент «Решение».** Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.



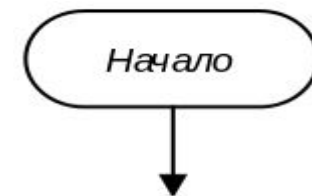
**Элемент «Цикл».** Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.



**Элемент «Соединитель».** Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.



**Элемент «Терминатор».** Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

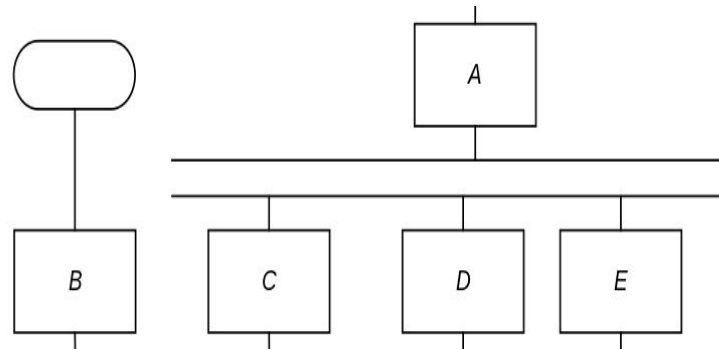


# Блок-схемы алгоритмов

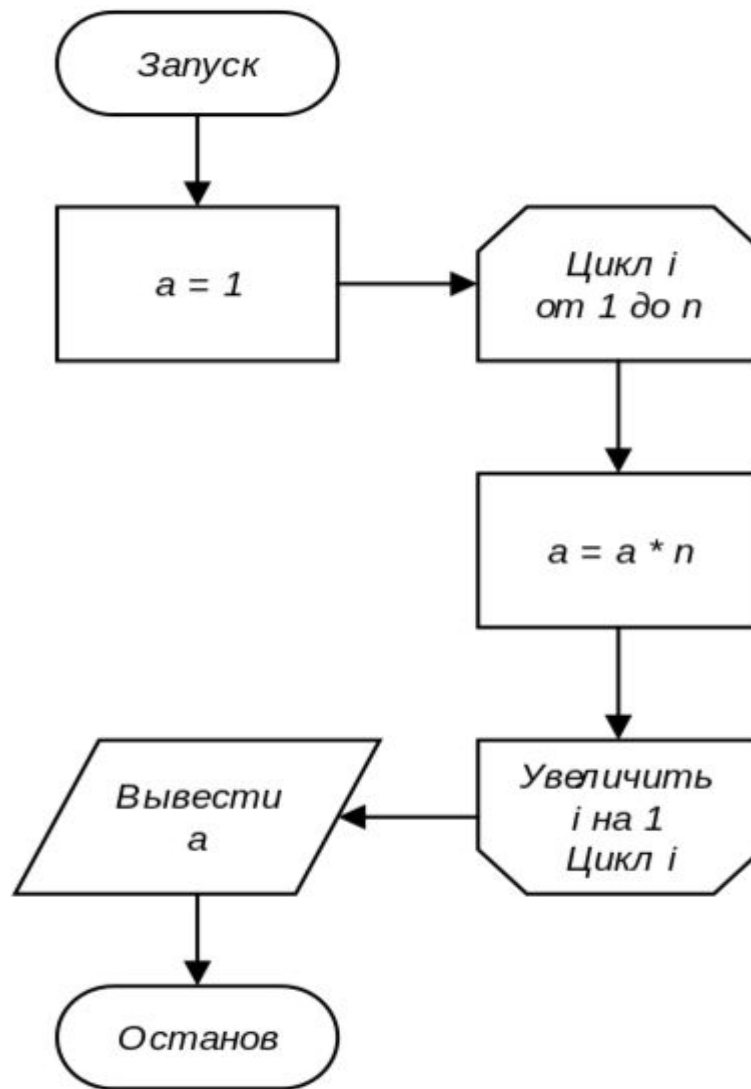
**Элемент «Комментарий».** Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры..

-- [ Входные параметры:  
массив `array[20]`;  
строка `str`.

**Элемент «Параллельные действия».** Символ представляется двумя параллельными линиями, отображает синхронизацию двух или более параллельных операций. В случае входа нескольких операций в параллельные линии, выполнение алгоритма будет продолжено только в случае окончания всех входящих процессов.



# Пример блок-схемы расчета факториала с использованием цикла



# Диаграммы UML (Unified Modeling Language)

**UML** — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

**UML** позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение, агрегация и поведение) и больше сконцентрироваться на проектировании и архитектуре.

# Диаграмма классов UML

