Глава 4. АЛГОРИТМЫ И СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

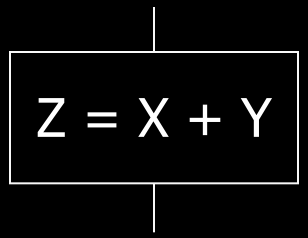
- Понятие и свойства алгоритма
- Язык блок-схем
- Простая программа, структурный подход к разработке алгоритмов
- Основные структуры алгоритмов
- Язык проектирования программ (псевдокод)
- Рекурсивные алгоритмы
- Алгоритмы поиска
- Алгоритмы сортировки
- Принципы объектно-ориентированного программирования

Алгоритм - формальное описание последовательности действий, которое необходимо выполнить для решения задачи.

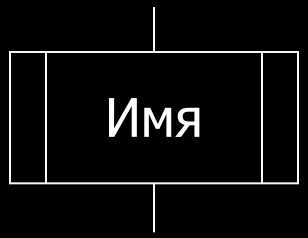
Свойства алгоритма

1. **Дискретность.** Алгоритм представляет процесс решения задачи как последовательность выполнения шагов-этапов. Для выполнения каждого эпата требуется определенное время, т.е. преобразование исходных данных в результат происходит дискретно во времени.
2. **Определенность (детерминированность).** Каждое правило алгоритма должно быть четким и однозначным. Отсюда выполнение алгоритма носит механический характер.
3. **Результативность (финитность, конечность).** Алгоритм должен приводить к решению задачи за конечное число шагов.
4. **Массовость.** Алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся исходными данными (*область применимости алгоритма*).

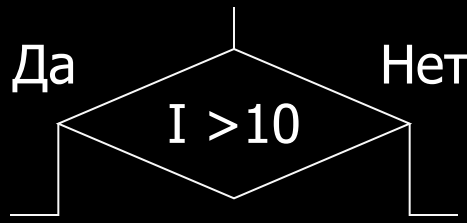
Язык **блок-схем** – способ формального описания алгоритмов



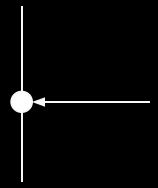
Обработка данных (вычисление, пересылка и т.п.)



Вызов процедуры



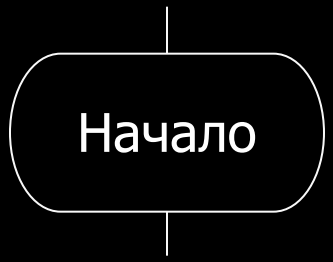
Проверка условия



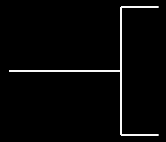
Соединительные линии и их объединение.

Ввод-вывод данных

Точки связи или соединители



Начало, завершение программы или подпрограммы



Комментарий

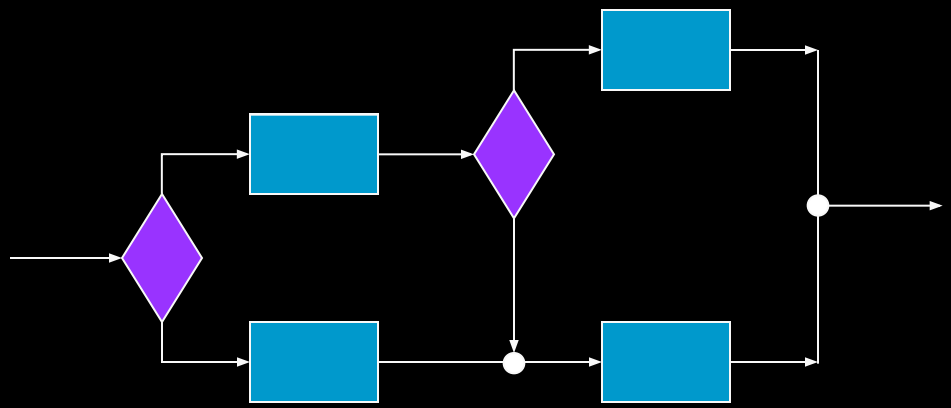
Основные (базовые) структуры алгоритмов – это ограниченный набор стандартных способов соединения отдельных блоков или структур блоков для выполнения типичных последовательностей действий.
Доказано, что программу для любой простой логической задачи можно составить из структур *следование*, *разветвление* и *повторение (цикл)*.

Эти базовые структуры были положены в основу технологии **структурного программирования**. Эта технология для разработки сложных программ рекомендует разбивать (декомпозировать) программу на подпрограммы (процедуры), решающие отдельные подзадачи, т.е. базируется на *процедурной декомпозиции*.

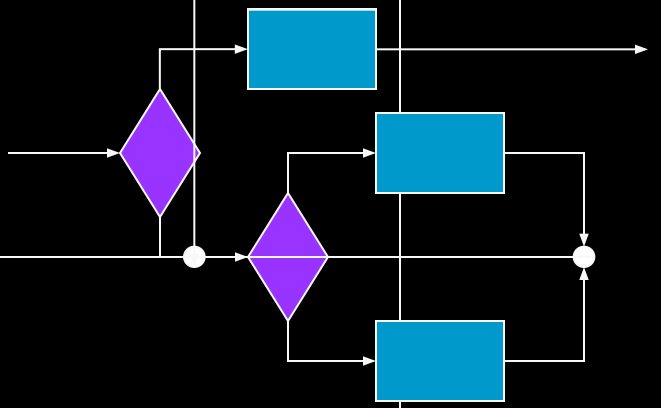
Простая программа - алгоритм, для которого:

- Существует единственный вход и единственный выход.
- Для каждого элемента алгоритма существует путь от входа к выходу через этот элемент (т.е. алгоритм не содержит бесконечных циклов и не содержит бесполезных (недостижимых) фрагментов).

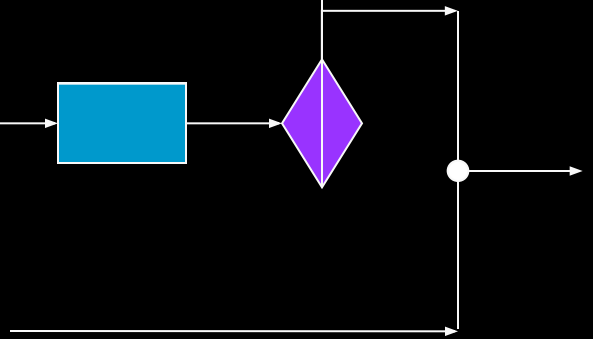
Примеры простой и непростых программ



Простая программа



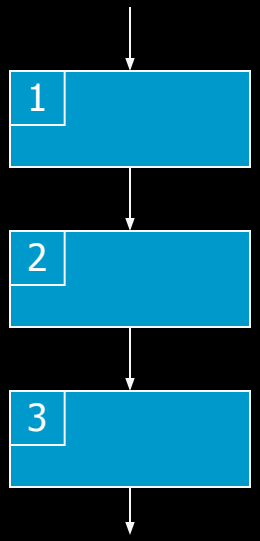
Бесконечный цикл



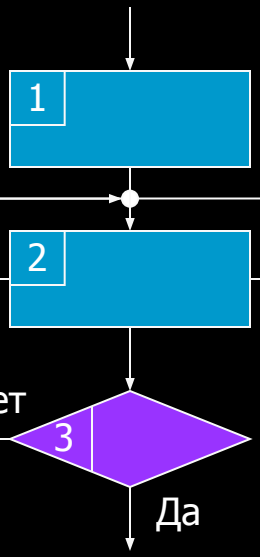
Недостижимый фрагмент

Основные (базовые) структуры алгоритмов и их производные

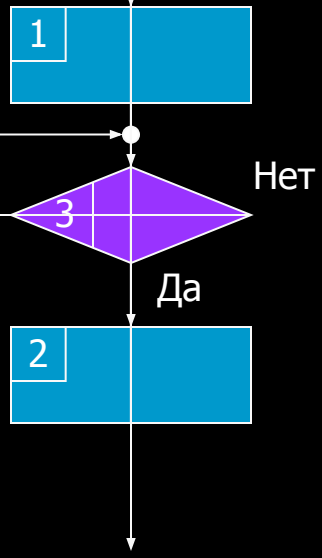
Следование - последовательное выполнение действий (блоков).



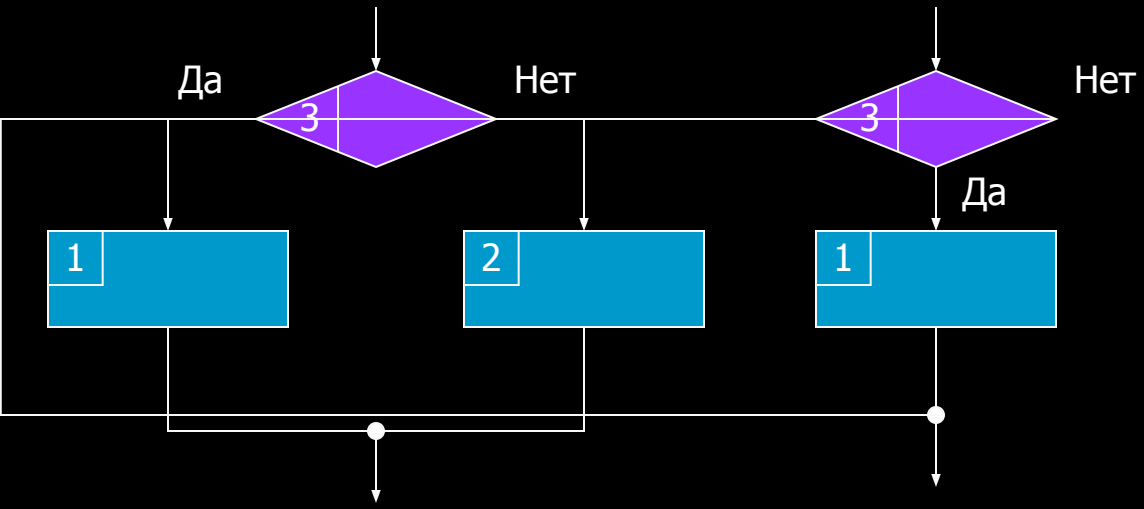
Цикл «До» (с постусловием) - тело цикла (блок 2) выполняется до тех пор, пока условие (блок 3) не станет истинным.

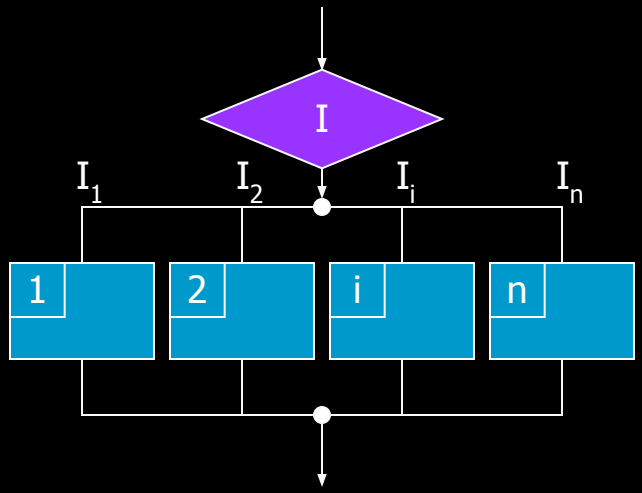


Цикл «Пока» (с предусловием) - пока не будет нарушено условие (блок 3), осуществляется повторение тела цикла (блок 2).



Основные (базовые) структуры алгоритмов и их производные





Разветвление - применяется, когда в зависимости от условия требуется выполнить либо одно действие, либо другое.

Обход - частный случай разветвления, когда одна ветвь не содержит ни каких действий.

Множественный выбор - обобщение разветвления, когда в зависимости от значения переменной I выполняется одно из нескольких действий.

Альтернативный способ описания логики программы на этапе проектирования – использование *псевдокода* (или языка проектирования программ PDL – Process Design Language). Он занимает промежуточное положение между естественным языком и языком программирования и состоит из внешнего синтаксиса и внутреннего синтаксиса.

Внешний синтаксис – заданный набор операторов, построенных по образцу языков программирования и описывающий логику программы. Внешний синтаксис соответствует основным структурам алгоритмов. Кроме того, к внешнему синтаксису также относятся процедуры и модули. **Процедура** – это хранимые в памяти машины подпрограммы, которые могут вызываться для выполнения из различных мест основной программы, либо из других процедур. Она вызывается и выполняется до завершения без сохранения внутренних данных. **Модуль** – это несколько процедур, организованных в систему для удобства работы пользователя. Модуль имеет доступ к общим данным, которые сохраняются между последовательными вызовами модуля.

Внутренний синтаксис – общий, обычно специально не определяемый синтаксис, пригодный для описания задач в данной области. Практически любое предложение, написанное на естественном языке, либо на специализированном языке (например, математические формулы) может быть использовано.

Операторы внешнего синтаксиса псевдокода

Следование. Записываются последовательно операции одна под другой. Для отделения части последовательности операторов используются операторы – do…end-do.

```
первая операция
вторая операция
do
    третья операция
end-do
```

Индексная последовательность (цикл по счетчику). Цикл с заранее определенным числом шагов (среднее между обычными последовательностью и классическим циклом).

```
for
    индексный список
do
    do-часть
end-do
```

Цикл-До. Операции структуры, включая модификацию until-теста, выполняются один или более раз до тех пор, пока until-тест не примет значение истина;

```
do
    do-часть
until
    until-тест
end-do
```

Операторы внешнего синтаксиса псевдокода

Цикл-Пока. Do-часть выполняется пока while-тест имеет значение истина. Do-часть модифицирует условие while-теста для того, чтобы окончить вычисления.

```
while
    while-тест
do
    do-часть
end-do
```

```
if
    if-тест
then
    then-часть
else
    else-часть
end-if
```

Разветвление.
Если…то…иначе.

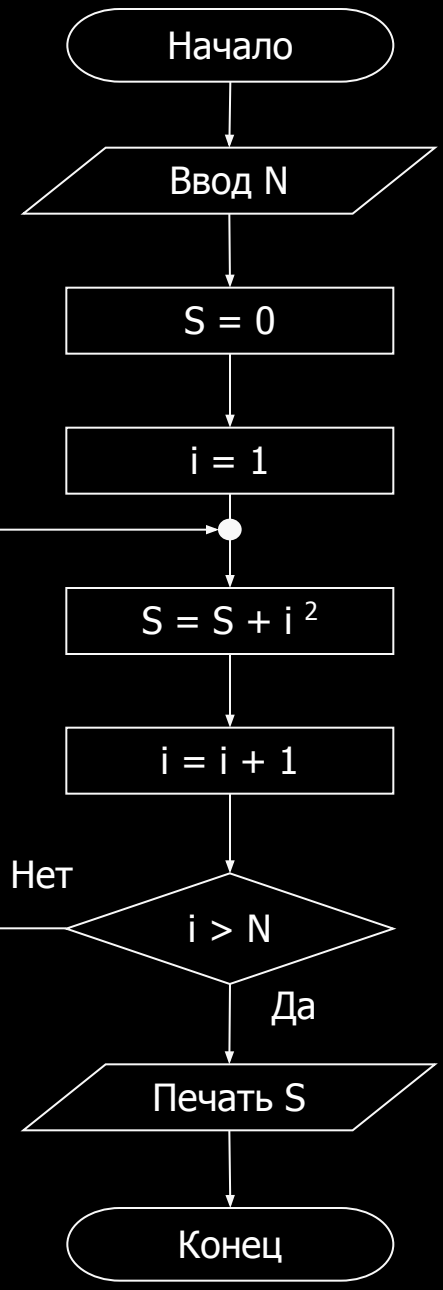
```
case
список 1
    case-часть 1
список 2
    case-часть 2
…
список n
    case-часть n
else
    else-часть
end-case
```

Множественный выбор.

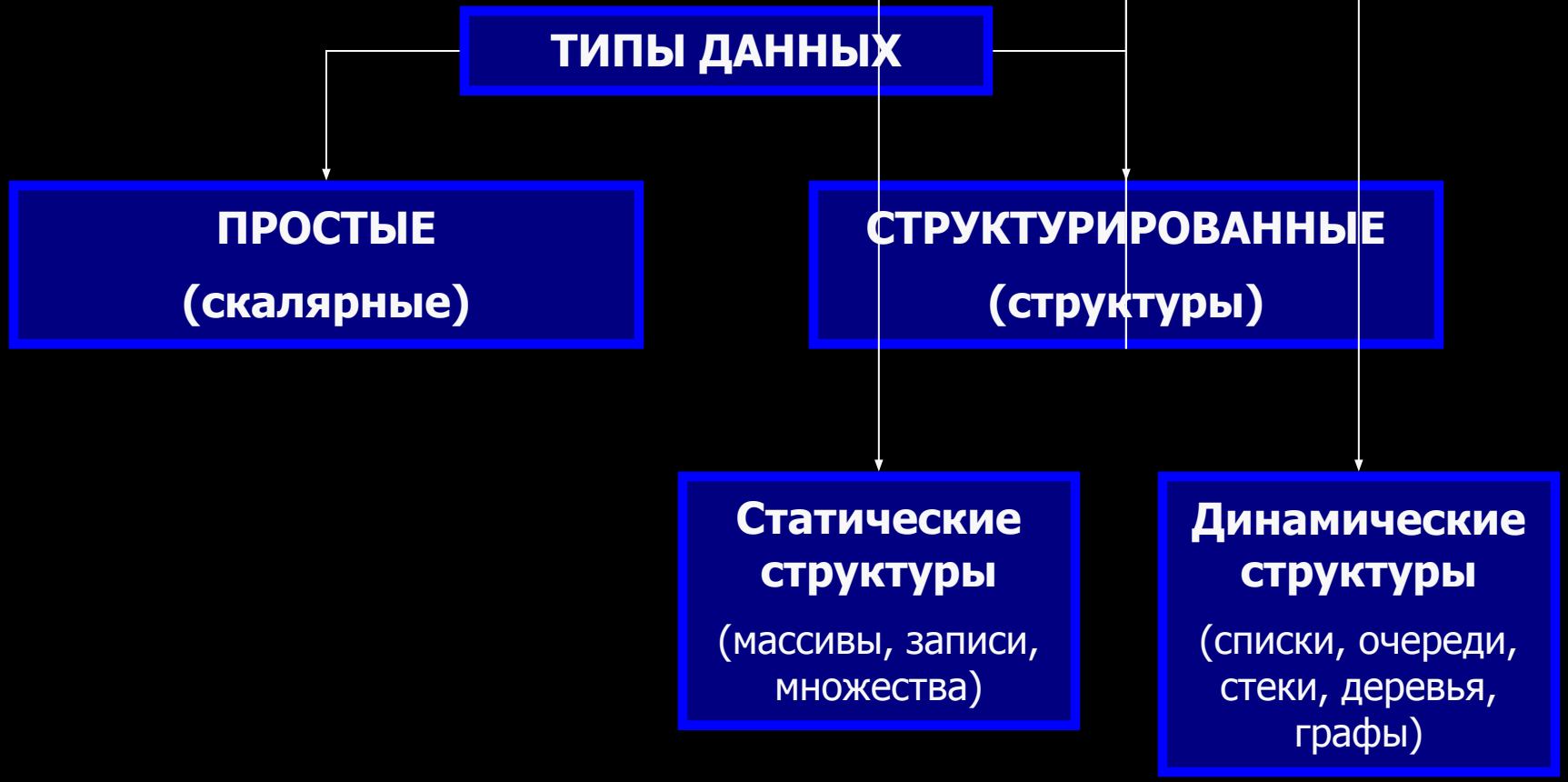
Алгоритмы вычисления суммы квадратов первых N целых чисел с использованием псевдокода и языка блок-схем

$$S = \sum_{i=1}^{N} i^2 = 1^2 + 2^2 + \ldots + N^2,$$

```
Ввести Число слагаемых
Сумма = 0
Номер = 1
do
    Сумма = Сумма + Номер²
    Увеличить Номер на единицу
until
    Номер > Число слагаемых
end-do
Напечатать Сумму
```



Помимо совокупности управляющих структур, важным аспектом структурного программирования является организация данных, участвующих в решении проблемы. Структура программы и строение данных неразрывно связаны.
«Программа – это конкретное, основанное на некотором реальном представлении и строении данных, воплощение абстрактного алгоритма». (Н. Вирт).



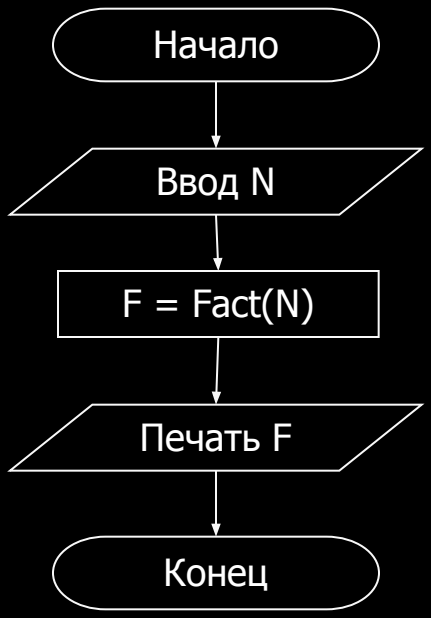
Рекурсия

Задача имеет *рекурсивное* решение, если его возможно сформулировать как известное преобразование другого, более простого решения той же задачи, хотя само решение (более простое) может быть неизвестно. Многократное повторение такого преобразования должно сходится к *базисному утверждению*.

Функция F является **рекурсивной**, если

1. F(N) = G(N,F(N-1)), где G известная функция;
2. F(1) – известно (базисное утверждение).

Алгоритм вычисления факториала N с использованием рекурсивной функции



Поиск

Поиск - обнаружение нужного элемента в некотором наборе (структуре) данных.
Элемент данных - это запись, состоящая из *ключа* (целое положительное число) и *тела*, содержащего некоторую информацию. Задача состоит в том, чтобы обнаружить запись с нужным ключом.

Линейный поиск.
Элементы проверяются последовательно, по одному, до тех пор, пока нужный элемент не будет найден. Для массива из N элементов требуется, в среднем, (N+1)/2 сравнений (вычислительная сложность $O_{ср}(N)$). Легко программируется, подходит для коротких массивов.

Двоичный (бинарный) поиск.
Применим, если массив заранее отсортирован (по возрастанию ключей). Ключ поиска сравнивается с ключом среднего элемента в массиве. Если значение ключа поиска больше, то та же самая операция повторяется для второй половины массива, если меньше - то для первой. Операция повторяется до нахождения нужного элемента. На каждом шаге диапазон элементов в поиске уменьшается вдвое. Требуется, в среднем, $(\log_2 N+1)/2$ сравнений (выч. сложность $O_{ср}(\log_2 N)$). Применяется для поиска (многократного) в больших массивах.

Сортировка

Сортировка (упорядочение) - переразмещение элементов данных в возрастающем или убывающем порядке. При выборе метода сортировки необходимо учитывать число сортируемых элементов (N) и до какой степени элементы уже отсортированы.

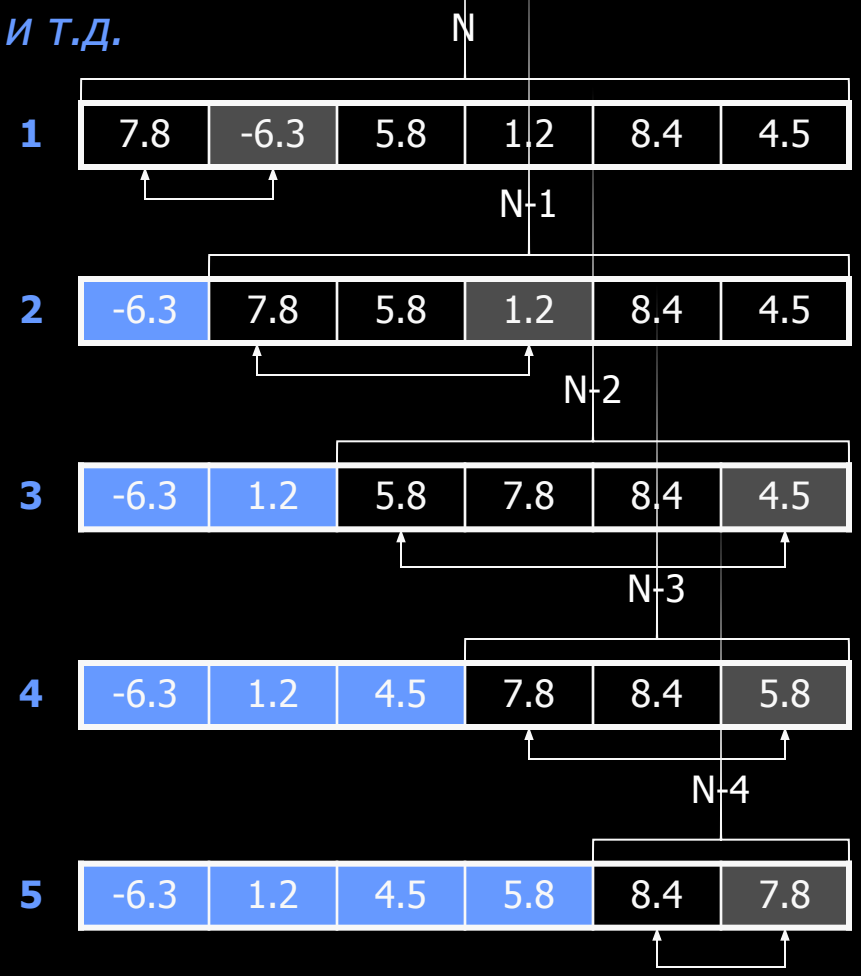
Критерии оценки метода сортировки:
- количество необходимых операций сравнения в зависимости от числа элементов N, вычислительная сложность алгоритма характеризуется с помощью O-функции, аргументом которой является другая функция от N;
- эффективность использования памяти

$$f = \frac{S(N)}{S(N) + \Delta S(N)},$$

где S(N) - объем памяти, занимаемый элементами данных до сортировки, ΔS(N) - объем дополнительной памяти, требуемой в процессе сортировки.

Сортировка методом выборки

Принцип: *Из массива выбирается наименьший элемент и помещается на место первого элемента массива, затем выбирается наименьший элемент из оставшихся и помещается во второй элемент массива и т.д.*

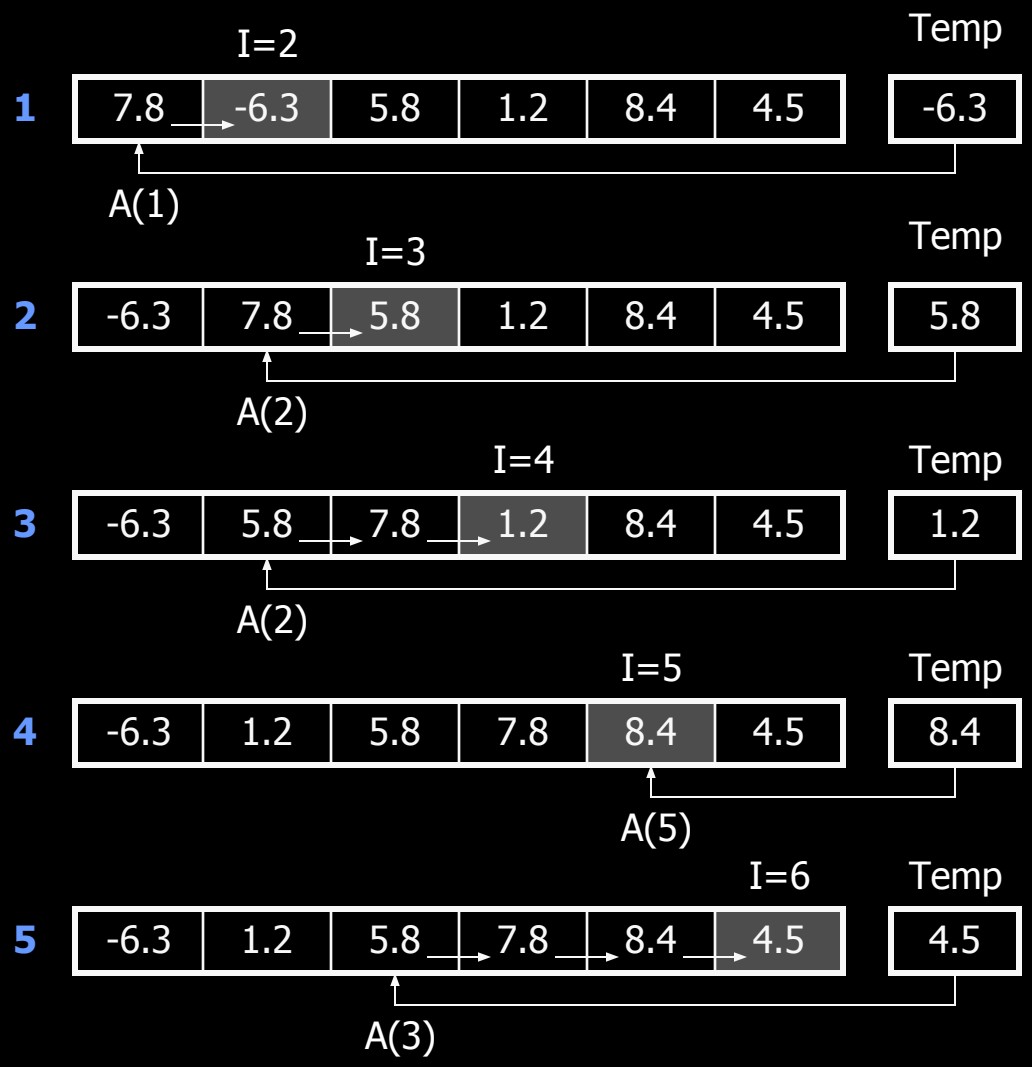



```
Ввести массив A(1..N)
for J = 1,N-1,1
do
    Мин.Эл. = A(J)
    Индекс Мин.Эл. = J
    for I = J+1,N,1
    do
        if A(I) < Мин.Эл.
        then
            Мин.Эл. = A(I)
            Индекс Мин.Эл. = I
        end-if
    end-do
    A(Индекс Мин.Эл.) = A(J)
    A(J) = Мин.Эл.
end-do
Вывести A(1..N)
```

Требуется, в среднем, $N(N+1)/2$ сравнений (выч. сложность $O(N^2)$, не зависит от начальной упорядоченности). Дополнительная память не нужна.

Сортировка включением

Принцип: *Элементы выбираются по очереди и помещаются в нужное место.*

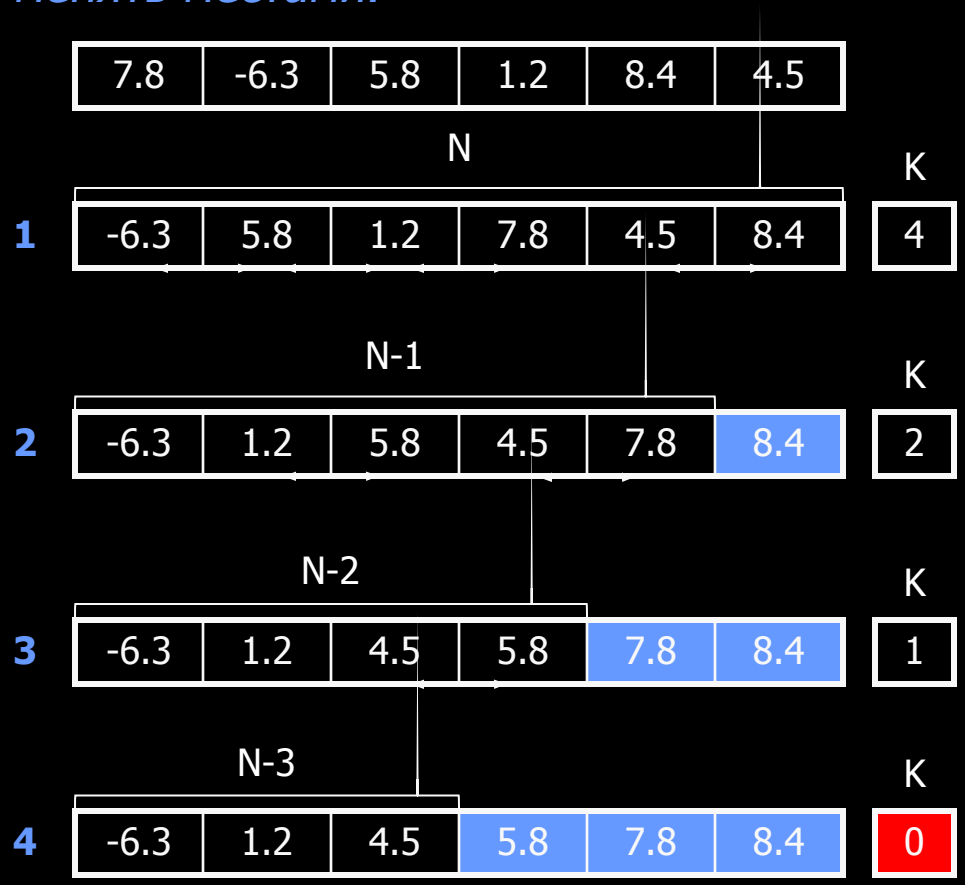



```
Ввести массив A(1..N)
for I = 2,N,1
do
    Temp = A(I)
    A(0) = Temp
    J = I-1
    while A(J) > Temp
    do
        A(J+1) = A(J)
        J = J-1
    end-do
    A(J+1) = Temp
end-do
Вывести A(1..N)
```

Требуется, в среднем, (N-1)(N/2+1)/2 сравнений (вычислительная сложность $O_{ср}(N^2)$). Скорость данного ме-тода зависит от начальной упо-рядоченности массива. Не тре-буется дополнительной памяти.

Сортировка обменами

Принцип: *Выбираются два элемента, и если друг по отношению к другу они не находятся нужном порядке, то меняются местами. Процесс продолжается пока никакие два элемента не нужно менять местами.*



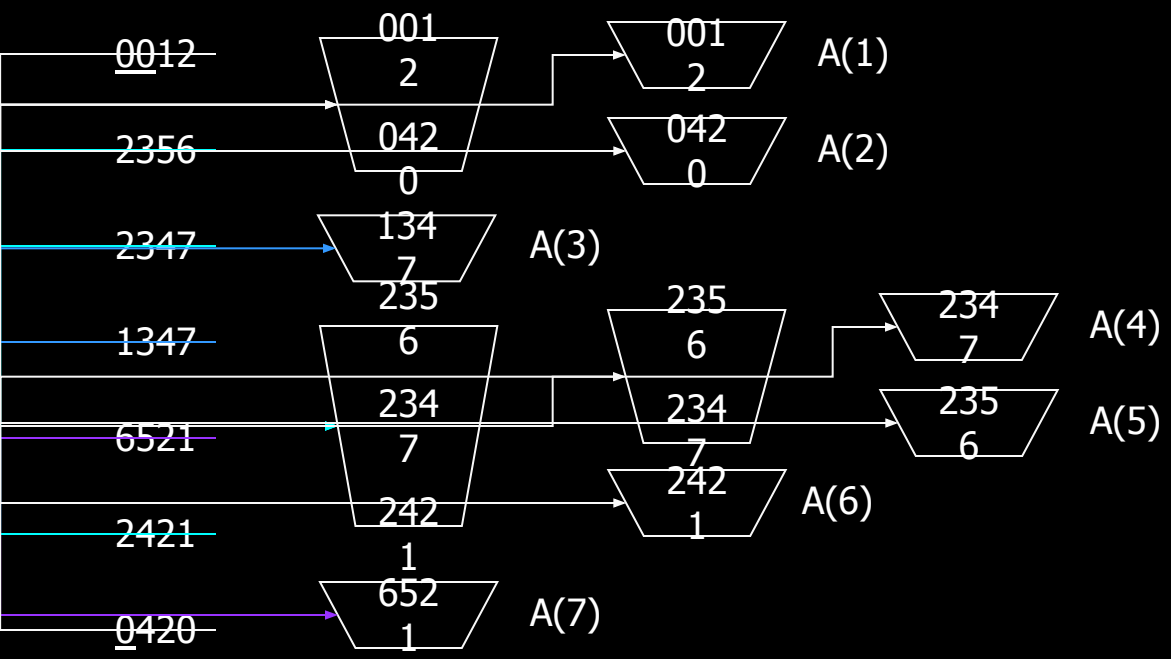

```
Ввести массив A(1..N)
K = 1, I = 1
while K <> 0
do
   K = 0
   for J = 1,N-I,1
   do
      if A(J) > A(J+1)
      then
         T = A(J),
         A(J) = A(J+1),
         A(J+1) = T, K = K+1
      end-if
   end-do
   I = I +1
end-do
Вывести A(1..N)
```

Вычислительная сложность данного метода сильно зависит от исходного расположения элементов. Минимальное значение числа сравнений – N-1 в полностью отсортированном массиве, максимальное – $(N^2-N)/2$ при начальной сортировке в обратном порядке. Средняя вычислительная сложность $O_{ср}(N^2)$. Доп. память не требуется.

Сортировка распределением (метод корзин)

Принцип: *Элементы массива рассматриваются как совокупность цифр (символов), первый шаг - сортировка по значению старшей цифры, затем полученные подмножества (группы) сортируются по значению следующей цифры и т.д.*

Каждый элемент массива A (1..N) - совокупность цифр $C_1C_2C_3…C_m$, где m - количество цифр максимального элемента (если какой-то элемент содержит меньше цифр, то он слева дополняется нулями).



Средняя вычислительная сложность $O_{ср}(N \log_2 N)$ и лучше, если m (число цифр) мало. Требуется дополнительный массив размером N, и еще массив размером 10, в котором подсчитывается число элементов с выделяемой цифрой 0,1,…9.

Быстрая сортировка

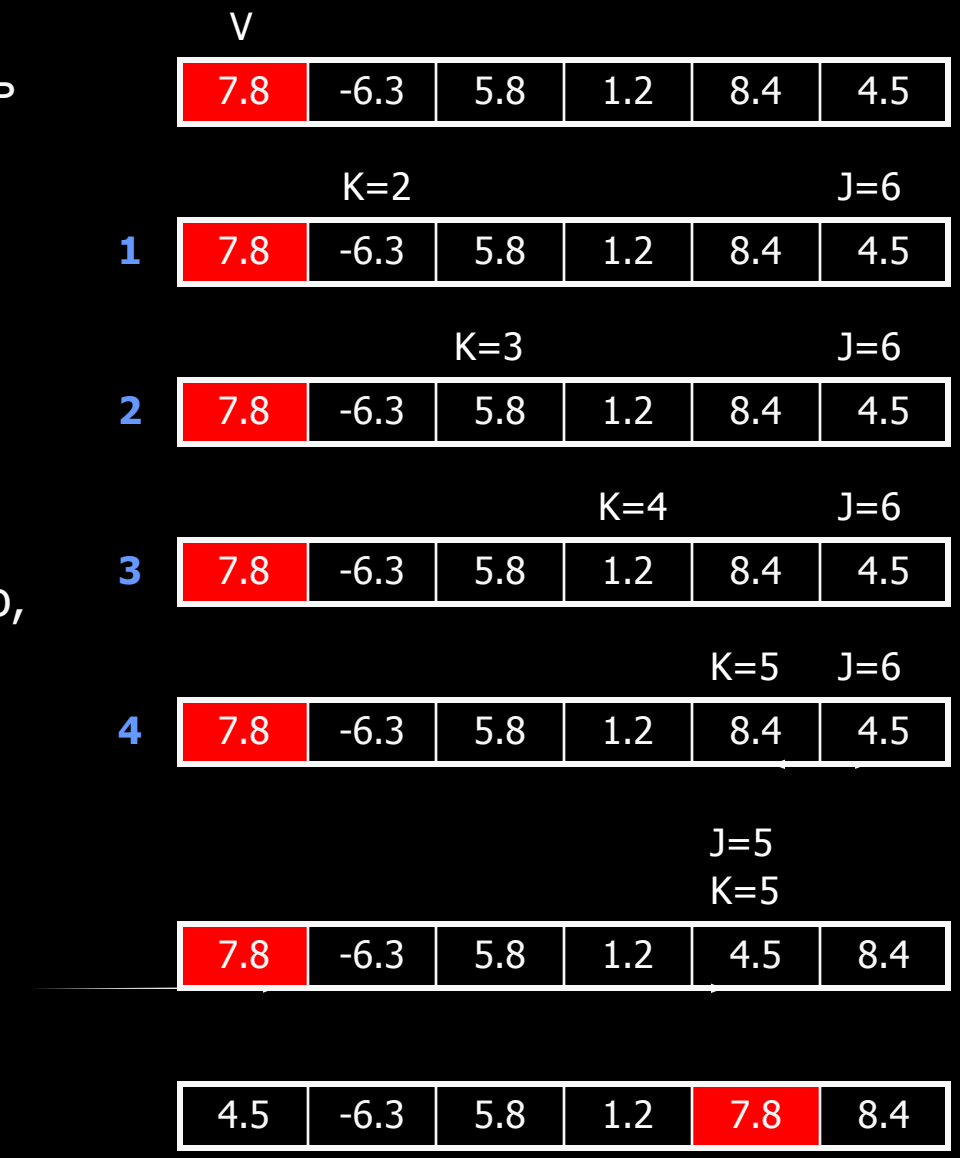
Принцип: *Определенным образом выделяется пороговый элемент. На первом этапе элементы обмениваются так, что новый массив оказывается разделенным пороговым элементом на две части: в левой все элементы меньше порогового, а в правой – больше или равны пороговому. Затем подобный способ используется для разделения каждого из новых массивов на две части и т.д.*

Алгоритм процедуры разбиения массива A(1..N) пороговым элементом, находящимся сначала на месте A(1).

```
{Процедура разбиения массива
A(1..N) пороговым элементом V}
V = A(1), K = 2, J = N
while K < J
do
   if A(K) < V
   then
      K = K + 1
   else
      if A(J) < V
      then
         Temp = A(K), A(K) = A(J)
         A(J) = Temp
      end-if
      J = J - 1
   end-if
end-do
if A(K) >= V
then
   K = K - 1
end-if
Temp=A(1),A(1)=A(K),A(K)=Temp
```

Быстрая сортировка

Средняя вычислительная сложность - $O_{ср}(N \log_2 N)$. Важное значение имеет выбор значения порогового элемента. В частности, если исходный массив близок к отсортированному, то при выборе пороговым элементом первого элемента (как в примере) вычислительная сложность алгоритма будет $O(N^2)$. Желательно, чтобы пороговый элемент в конечном итоге разделил массив приблизительно на две равные части.



Сортировка слиянием

Принцип: *Два отсортированных массива соединяются в один массив таким образом, чтобы и он стал отсортированным.*

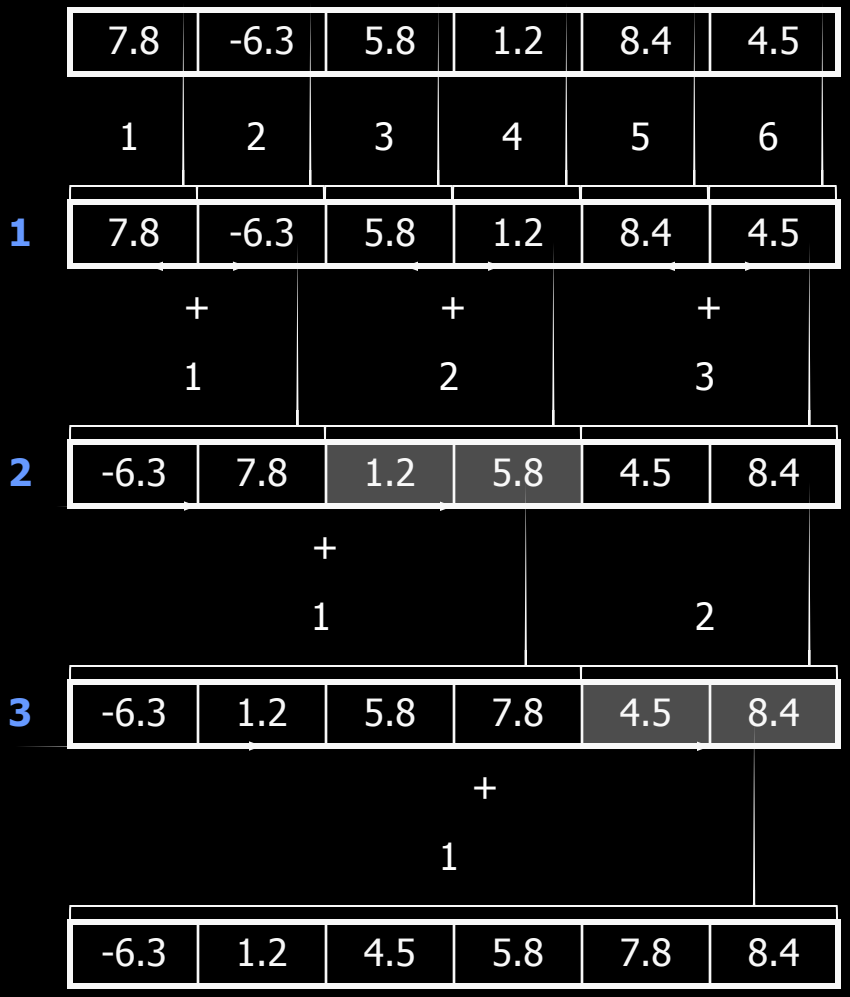
Алгоритм слияния отсортированных массивов B(1..M) и C(1..L) в массив A(1..M+L) заключается в следующем. В качестве A(1) выбираем наименьший из B(1) и C(1). Если это B(1), то в качестве A(2) - наименьший из B(2) и C(1) и т.д.

```
Ввести массивы B(1..M),C(1..L)
I = 1, J = 1
for K = 1,M+L,1
do
   if I > M
   then
      A(K) = C(J), J = J + 1
   else
      if J > L
      then
         A(K) = B(I), I = I + 1
      else
         if B(I) < C(J)
         then
            A(K) = B(I),I = I + 1
         else
            A(K) = C(J),J = J + 1
         end-if
      end-if
   end-if
end-do
Вывести A(1..K)
```

Сортировка слиянием

Если имеется один неотсортированный массив A(1..N), то его можно рассматривать как совокупность N отсортированных массивов, каждый из которых состоит из одного элемента. Первый шаг - слияние массивов попарно, затем объединение пар в четверки и т.д.

Средняя вычислительная сложность алгоритма - $O_{ср}(N \log_2 N)$. Требуется дополнительный массив, содержащий N элементов.



Особенности объектно-ориентированной программирования

Структурное программирование приспособлено для описания *действий*, а ООП - для описания *моделей*.

Абстракция – это суждение или представление о некотором объекте, которое содержит только свойства, являющиеся существенными в данном контексте. Абстракция – эффективное средство против сложности программирования, поскольку оно позволяет программисту сосредоточиться на существенных свойствах объекта и игнорировать менее важные свойства.

Объектный стиль программирования связан с воздействием на объекты (говорят, с передачей объекту *сообщений*). При этом проектирование системы базируется на том условии, что никакая подсистема данного уровня не должна зависеть от устройства любой другой подсистемы этого уровня (взаимодействуют, но не зависят).
Инкапсуляция – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; в частности, это скрытие переменных (полей) объекта с целью обеспечения доступа к ним только посредством методов класса. Идея инкапсуляции, в частности, реализована в модуле при его разделении на секции интерфейса и реализации.

Значительно упростить понимание сложных задач удается за счет *усложнения иерархии*. Усложнение иерархии от уровня к уровню достигается за счет наследования. **Наследование** – это расширение свойств наследника за счет принятия всех свойств предка.

ООП использует также понятие **"полиморфизма"**, состоящее в возможности переопределять методы класса-родителя. Иными словами, полиморфизм означает общий род действий, которые могут быть выполнены многими специфическими путями, в зависимости от того, какие объекты выполняют действия. Неразрывно с полиморфизмом связано понятие **динамического связывания** – возможность отложить подстановку реального адреса некоторой подпрограммы-метода с этапа компиляции (раннее связывание) до момента выполнения программы (позднее связывание), что используется для стандартизации работы с объектами.

Любой объект в ООП обладает определенным *поведением* и *состоянием*. Поведение объекта – это то, как он реагирует на воздействия. Состояние объекта представляет собой суммарный результат его поведения. Сообщение, посланное объекту, активизирует совокупность операций над объектом, которая называется *методом*. Задавая структуру обмена сообщениями между объектами, программист получает совокупность операций, которые и составляют программу.

Основные типы операций над объектами

Название	Содержание
Конструктор	Создает и инициализирует объект.
Деструктор	Освобождает объект, т.е. разрушает его.
Модификатор	Изменяет состояние объекта.
Селектор	Считывает состояние объекта без изменения этого состояния.
Итератор	Организует доступ ко всем частям объекта в строго определенной последовательности.

Внутренность объекта

ОКРУЖНОСТЬ А

Переменные

- **Позиция 15,20**
- **Размер 5**

Методы

- **Форма**
  - **Маркер** ПероВверх
  - **Маркер** СдвигК : Позиция
  - **Маркер** СдвигНа : Размер
  - **Маркер** ПероВниз
  - **Маркер** ЧертитьДугу : Размер,360
- **Высветить**
  - **Маркер** Черный
  - **Себе** Форма
- **Стереть**
  - **Маркер** Белый
  - **Себе** Форма

МАРКЕР

Методы

- **ПероВверх**
- **ПероВниз**
- **Черный**
- **Белый**
- **СдвигК : X,Y**
- **СдвигНа : Z**
- **ЧертитьДугу : R,ф**
- **…**

Сообщения (операции) объекту ОкружностьА:

```
ОкружностьА.Высветить

ОкружностьА.Стереть
```

Иерархия классов