



Сущность понятия «Сложность
алгоритма»



Основные понятия

- Алгоритм — набор инструкций описывающих порядок (последовательность) действий исполнителя для достижения результата решения задачи за конечное число действий.

- Алгоритм решения вычислительной задачи представляет собой совокупность правил преобразования исходных данных в результатные.



Свойства алгоритма

- **Детерминированность (определенность).** Предполагает получение однозначного результата вычислительного процесса при заданных исходных данных. Благодаря этому свойству процесс выполнения алгоритма носит механический характер;
- **Результативность.** Указывает на наличие таких исходных данных, для которых реализуемый по заданному алгоритму вычислительный процесс должен через конечное число шагов остановиться и выдать искомый результат;

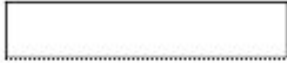





Свойства алгоритма

- **Массовость.** Это свойство предполагает, что алгоритм должен быть пригоден для решения всех задач данного типа;
- **Дискретность.** Означает раздленность определяемого алгоритмом вычислительного процесса на отдельные этапы, возможность выполнения которых исполнителем (компьютером) не вызывает сомнений.



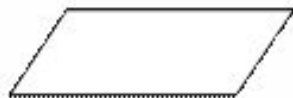
Графическое представление алгоритмов

| Название символа | Обозначение и пример заполнения | Пояснение |
|--------------------------|--|---|
| Процесс |  | Вычислительное действие или последовательность действий |
| Решение |  | Проверка условий |
| Модификация |  | Начало цикла |
| Предопределенный процесс |  | Вычисления по стандартной подпрограмме |



Графическое представление алгоритмов

Ввод-вывод



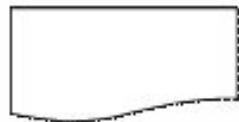
Ввод-вывод в общем виде

Пуск-останов



Начало, конец алгоритма, вход и выход в подпрограмму

Документ

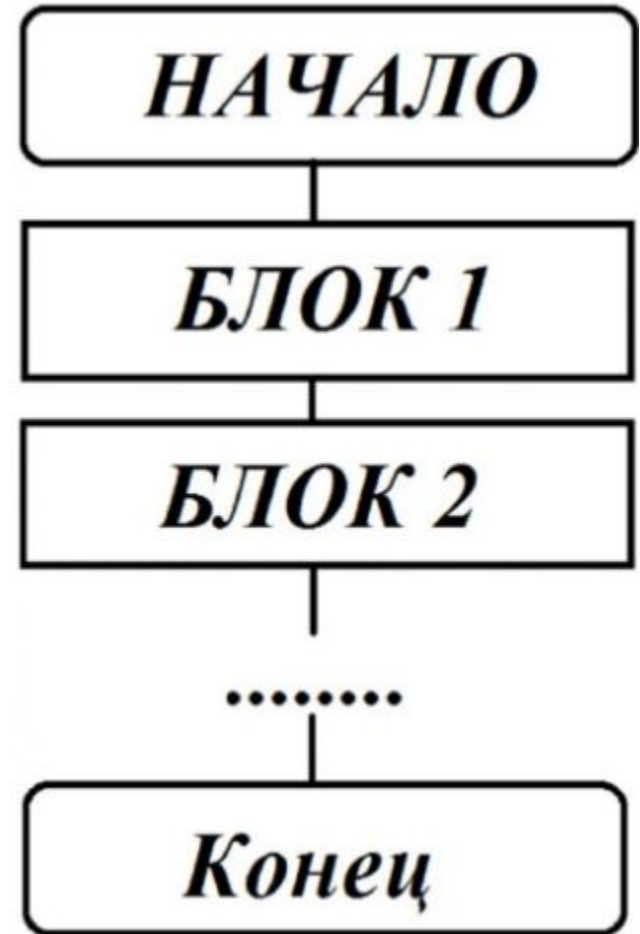


Вывод результатов на печать



Линейный тип алгоритмов

- Это самый простой вид, который состоит из определенной последовательности действий, они не зависят от того, какие данные вписаны изначально. Есть несколько команд, которые выполняются однократно и только после того, как будет сделана предшествующая. Линейная блок-схема выглядит таким образом:



Разветвляющиеся алгоритмы

- Разветвляющийся алгоритм – это процесс, в котором дальнейшее действие зависит от того, как выполняется условие и какое получается решение. Каждое направление действия – это ветвь.



Обход



Разветвление



Множественный выбор



Разветвляющиеся алгоритмы

- Разветвляющийся алгоритм – это процесс, в котором дальнейшее действие зависит от того, как выполняется условие и какое получается решение. Каждое направление действия – это ветвь.



Разветвление



Множественный выбор



Циклический алгоритм

- Алгоритм, в котором многократно повторяются однотипные вычисления. По определению, цикл – это определенная последовательность каких-либо действий, выполняемая многократно (более, чем один раз)
 - У которых известно число повторений действий (их еще называют циклами со счетчиком).
 - У которых число повторений неизвестно – с постусловием и предусловием.



Циклы со счетчиками

- Такой тип алгоритмов показывает, что заранее известно количество повторений данного цикла. И это число фиксировано. При этом переменная, считающая число шагов (повторений), так и называется – счетчик.

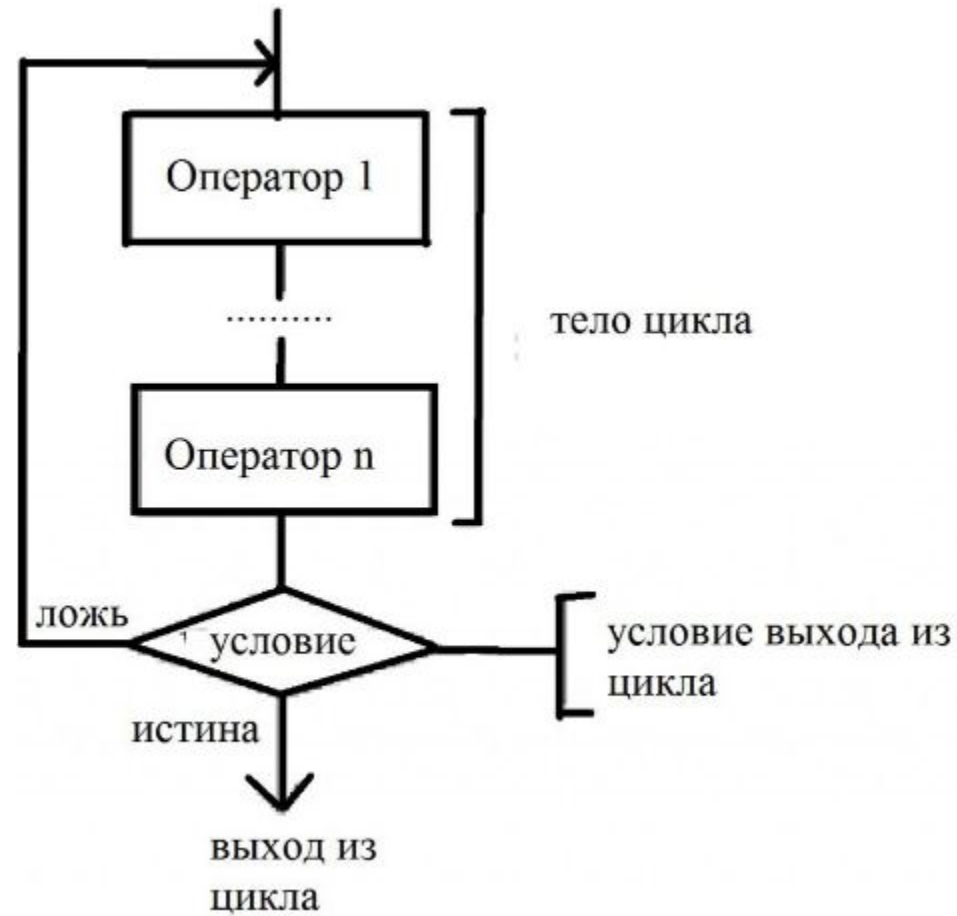


Циклы с условиями

- Цикл с предусловием – это тип алгоритма, в котором непосредственно перед началом выполнения тела осуществляется проверка условия, при котором допускается переход к следующему действию. Обратите внимание на то, как изображаются элементы блок-схемы.
- Цикл с постусловием – особенность данного алгоритма заключается в том, что неизвестно заранее число повторений. А условие задается уже после того, как произошел выход из тела. Отсюда видно, что тело, независимо от решения, будет выполняться как минимум один раз.



Циклы с условиями



Объектно-ориентированное программирование

- **Инкапсуляция** — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.
- **Наследование** — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствуемой функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом
- **Полиморфизм** — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.



Инкапсуляция (виды модификаторов)

- ▣ **Public** – уровень предполагает доступ к компоненту с этим модификатором из экземпляра любого класса и любого пакета.
 - ▣ **Protected** – уровень предполагает доступ к компоненту с этим модификатором из экземпляров родного класса и классов-потомков, независимо от того, в каком пакете они находятся.
 - ▣ **Default** – уровень предполагает доступ к компоненту с этим модификатором из экземпляров любых классов, находящихся в одном пакете с этим классом.
 - ▣ **Private** – уровень предполагает доступ к компоненту с этим модификатором только из этого класса.
-



Полиморфизм

□ *“один интерфейс, множество методов”.*

□ `public class Parent {
int a = 2;}`

□ `public class Child extends Parent {
int a = 3;}`

□ `Child c = new Child();
System.out.println(c.a);`

□ `Parent p = c;
System.out.println(p.a);`



Базовый синтаксис Java

- **Имя файла всегда идентично имени класса**
- Символы чувствительны к регистру (даже в Windows);
- **Обработка всегда начинается в main**
- **public static void main (String[] args);**
- Обычно процедуры называются «методами», а не «функциями»;
- **Вывод осуществляется с помощью System.out**

```
1 public class HelloWorld HelloWorld {
2     public static void main (String [] args) {
3         System.out.println («Hello, world.»);
4     }
5 }
```



Объявление переменных

- ▣ **int** x; // Объявление целочисленной переменной x
- ▣ **double** a, b; // Объявление двух вещественных переменных a и b
- ▣ **char** letter = 'Z'; // Объявление символьной переменной letter, инициализация начальным значением 'Z'
- ▣ **boolean** b1 = true, b3 = false; // Объявление трех логических переменных, первая из них будет иметь значение true, последняя — false



Основные операции

□ Математические операции

| Операция | Использование | Описание |
|----------|---------------|---|
| + | $op1 + op2$ | Складывает $op1$ и $op2$ |
| - | $op1 - op2$ | Вычитает $op1$ из $op2$ |
| * | $op1 * op2$ | Умножает $op1$ на $op2$ |
| / | $op1 / op2$ | Делит $op1$ на $op2$ |
| % | $op1 \% op2$ | Вычисляет остаток от деления $op1$ на $op2$ |

□ Операции сравнения

| Операция | Использование | Возвращает истину(true), если |
|----------|---------------|-------------------------------|
| > | $op1 > op2$ | $op1$ больше чем $op2$ |
| >= | $op1 >= op2$ | $op1$ больше или равен $op2$ |
| < | $op1 < op2$ | $op1$ меньше $op2$ |
| <= | $op1 <= op2$ | $op1$ меньше или равно $op2$ |
| == | $op1 == op2$ | $op1$ и $op2$ равны |
| != | $op1 != op2$ | $op1$ и $op2$ не равны |

□ Логические операции

| Операция | Использование | Возвращает истину(true), если |
|----------|------------------|--|
| && | $op1 \&\& op2$ | $op1$ и $op2$ оба истинны (конъюнкция) |
| | $op1 \ \ op2$ | один из $op1$ или $op2$ истинен (дизъюнкция) |
| ! | $!op$ | op — ложь (отрицание) |
| ^ | $op1 \wedge op2$ | $op1$ и $op2$ различны (исключающее или) |

Операторы цикла

□ **while** (условие) команда

□ Пример:

```
int x = 2;
```

```
while (x <= 10)
```

```
{System.out.println(x);x += 2;}
```

□ **for** (команда инициализации; условие; команда перехода) тело_цикла

```
for (int i = 1; i <= 10; i++) тело_цикла;
```

□ Пример

```
for (int i = 1; i <= 5; i++) System.out.println(i*2);
```

