Требования к алгоритму, формы его представления

Валов Андрей Михайлович, к.п.н.,
доц. каф. ИТвО НИПКиПРО

Федеральный Государственный Образовательный СТАНДАРТ

Алгоритм и исполнитель

- **Алгоритм**— точный набор инструкций, описывающих **порядок** действий исполнителя для достижения результата решения задачи за конечное время.
- **Исполнитель алгоритма** — это некоторая абстрактная или реальная (техническая, биологическая или биотехническая) система, способная выполнить действия, предписываемые алгоритмом.

Особенности исполнителей

Исполнителя характеризуют:

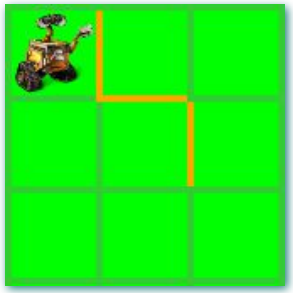
- среда деятельности (обстановка)
- система команд управления (**СКИ***)
- элементарные действия (в ответ на команды)
- режимы управления (прямое, программное)
- отказы (ответ на вызов команды в недопустимом для выполнения состоянии среды)

***Система команд исполнителя**:

- конечное множество инструкций, которые понимает и может выполнить исполнитель.

Пример

Исполнитель «Простой Робот»



- среда деятельности – *клетчатое поле произвольных размеров, на поле могут находиться стены*
- система команд управления (**СКИ**) – *ВЛЕВО, ВПРАВО, ВВЕРХ, ВНИЗ*
- элементарные действия – *на команду ВЛЕВО смещается влево на 1 клетку, и т.п.*
- режимы управления - *программное*
- отказы – *при попытке пройти сквозь стену происходит отказ*

Язык

- **Язык** – определенная система символьного представления информации; множество символов и совокупность правил, определяющих способы составления из этих символов осмысленных сообщений.
- Языки разделяют на **естественные**, **формальные** и **промежуточные** *(например, соотв. русский, алгоритмический и латынь)*

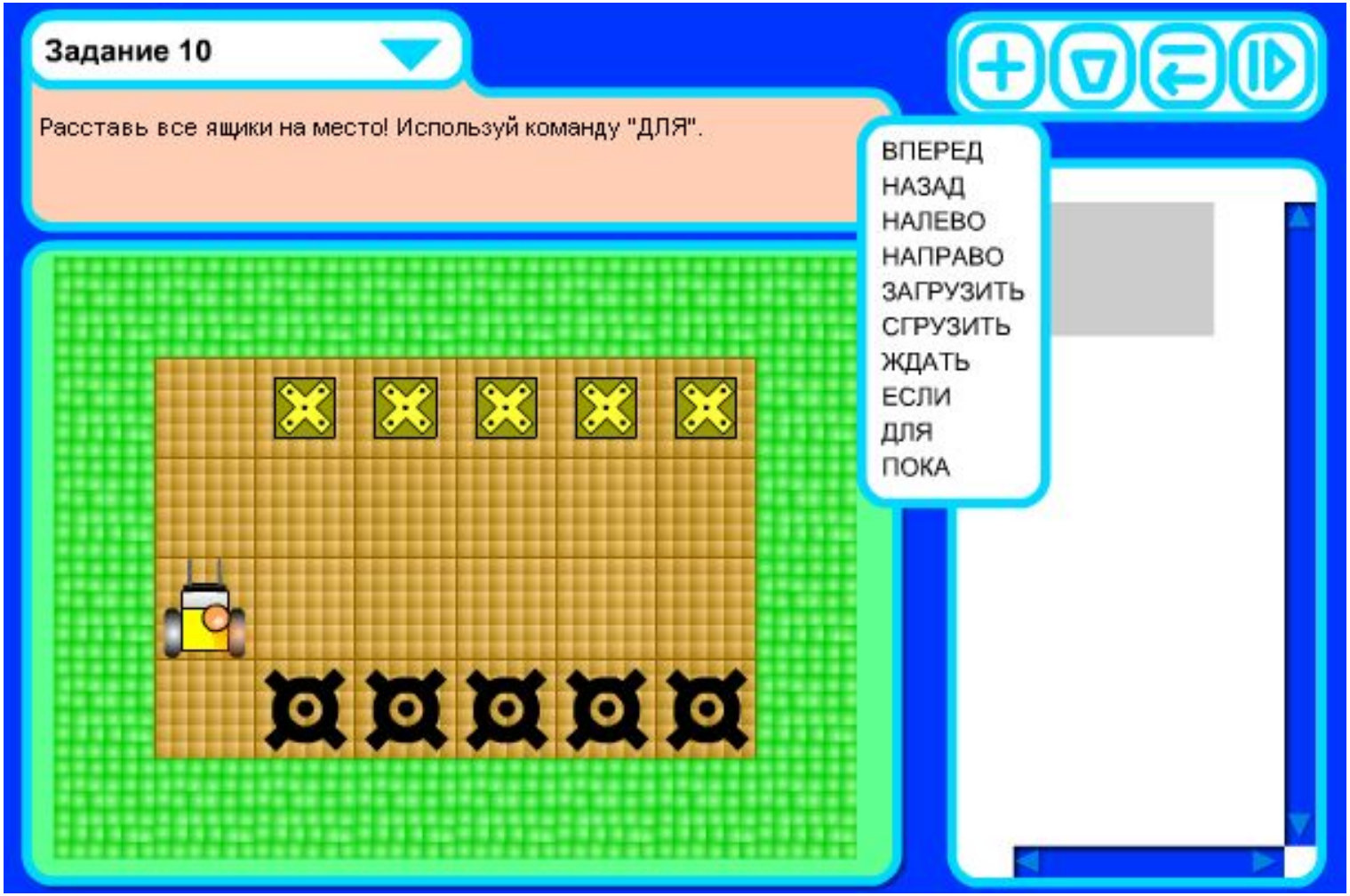
Алфавит, словарь, алгоритмический язык и программа

- Множество символов, используемое в языке, образует **алфавит**.
- Множество слов языка образует **словарь.**
- **Алгоритмический язык (АЯ)** – система обозначений и правил для единообразной и точной записи и исполнения алгоритмов.
- Алгоритм, записанный на языке **программирования**, становится **программой.**

Словарь языка исполнителя

- Алгоритмический язык имеет свой **словарь.**
- Основу словаря составляют слова, употребляемые для записи **команд**, входящих в СКИ.
- Также в словарь входят специальные **служебные слова** – *например, элементы составных алгоритмических конструкций типа ветвление, цикл и др.*

Пример словаря исполнителя «Погрузчик»

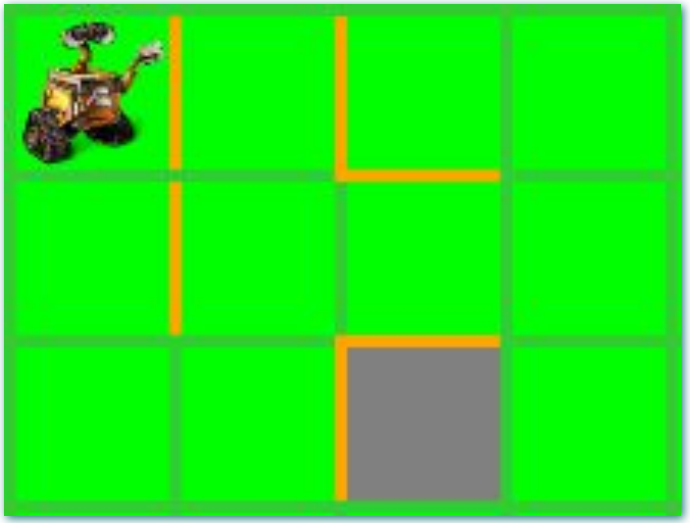


Как измерить объем информации, содержащийся в линейном алгоритме?

- Измеряем не объем информации в тексте алгоритма, а объем самого алгоритма!
- Значит, важно учесть число команд, выданных исполнителю, а не число символов в команде.
- Например, исполнитель «Простой Робот» понимает 4 вида команд: ВПРАВО, ВЛЕВО, ВВЕРХ, ВНИЗ.
- Для записи одной команды достаточно 2 бит, т. к. ими можно закодировать все 4 команды – 00, 01, 10, 11.
- Подсчитав число команд в алгоритме, можно узнать объем информации, который он содержит!

Пример задачи

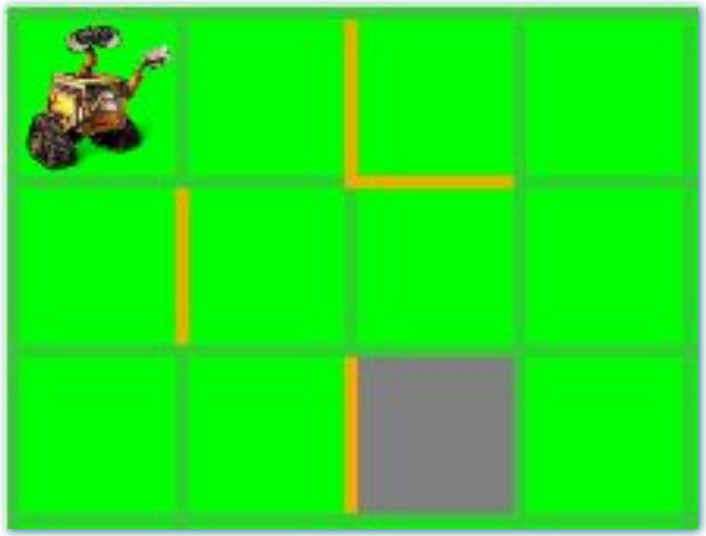
- Какой объем информации (в битах) будет содержать алгоритм кратчайшего перехода «Простого Робота» в клетку, помеченную серым цветом?



Важный нюанс!

- Для решения отдельно взятой задачи может быть использован **только часть** арсенала СКИ исполнителя.
- Тогда путем «отброса» ненужных команд можно уменьшить информационный «вес» оставшихся команд:
  - ВПРАВО – 0
  - ВНИЗ – 1

Вместо 8 бит алгоритм кратчайшего перехода будет содержать … бит.



«Классические» отечественные учебные исполнители и их среды

- **Кузнечик** – работа с числовой прямой
- **Чертежник** – работа в декартовых координатах
- **Черепашка** – работа в полярных координатах
- **Таракан** или **Муравей** – работа со строками и символами
- **Кладовщик** – работа с таблицами
- **Директор строительства** – параллельное исполнение алгоритмов

Требования к алгоритму

- **Дискретность** – достижение результата путем выполнения набора простых действий - 1 шаг за 1 промежуток времени.
- **Детерминированность** (определенность) - алгоритм выдает один и тот же результат для одинаковых исходных данных.
- **Понятность** – алгоритм включает только команды из СКИ.

Требования к алгоритму

- **Конечность** – завершение работы за конечное число шагов (при верно заданных исходных данных).
- **Массовость** - алгоритм применим для разного набора исходных данных.
- **Результативность** – завершение работы конкретным результатом.

Примеры задач

- Исполнитель должен приготовить сладкий зеленый чай, для чего составили алгоритм:

взять стакан

налить чай

подать стакан

Дискретность
Детерминированность
Понятность

Какие требования к алгоритму могли быть нарушены?

Примеры задач

- Исполнитель имеет в СКИ две команды: **увеличить на 1** и **уменьшить на 3**. Для него составлен алгоритм:

увеличить на 1
увеличить на 1
умножить на 3

Понятность

Какое требование к алгоритму нарушено?

Примеры задач

- Исполнитель при выполнении команды **вперед** совершает от 1 до 3 шагов.

пока (справа_свободно) {вперед}

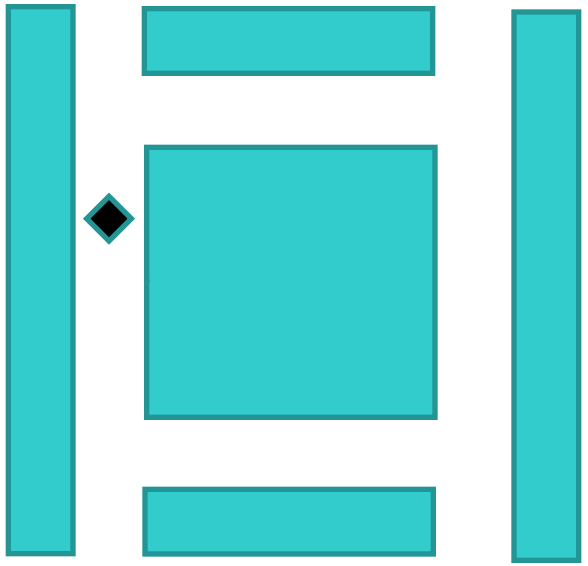
Детерминированность

Какое требование к алгоритму нарушено?

Примеры задач

- Исполнитель в лабиринте выполняет алгоритм, предназначенный для выхода из лабиринта:

пока (справа_стена)
{вперед}
иначе
{направо;вперед}



Конечность

Какое требование к алгоритму нарушено?

Варианты постановки задач (от простого – к сложному)

- Зная СКИ, выполнить роль исполнителя (формально исполнить составленный алгоритм).
- В рамках заданных СКИ и обстановки построить (записать) алгоритм решения поставленной задачи.
- Определить нужный набор исходных данных и построить алгоритм для решения задачи при помощи исполнителя.
- Определить исполнителя и его СКИ в рамках заданной обстановки, построить для него алгоритм.

Ошибки в алгоритмах

- Алгоритм **содержит ошибки**, если приводит к получению неправильных результатов либо не даёт результатов вовсе.
- Алгоритм **не содержит ошибок**, если он даёт правильные результаты для любых допустимых исходных данных.

Ошибки в алгоритмах

- При несоблюдении правил записи алгоритма на АЯ возникают **синтаксические ошибки**. Их легко обнаружить, просто сравнивая код со словарем.
- Вызов команды исполнителя в ситуации, когда он не может ее выполнить, приводит к отказу – это **семантическая ошибка**.
- **Логические ошибки** находить гораздо сложнее, для этого приходится привлекать специальные **средства отладки** – точки останова, ведение лог-файла и др.

Пример задач

- Для учебного исполнителя **Счетовод** составили линейный алгоритм получения из числа **0** числа **12**:

увеличить на 1

умножить на 3

увиличить на 1

помножить на 3

Синтаксическая

Какой тип ошибки допущен в алгоритме ?

Пример задач

- Для учебного исполнителя **Калькулятор** составили алгоритм:

увеличить на 1
умножить на 3
разделить на 0

Семантическая

Какой тип ошибки допущен в алгоритме ?

Пример задач

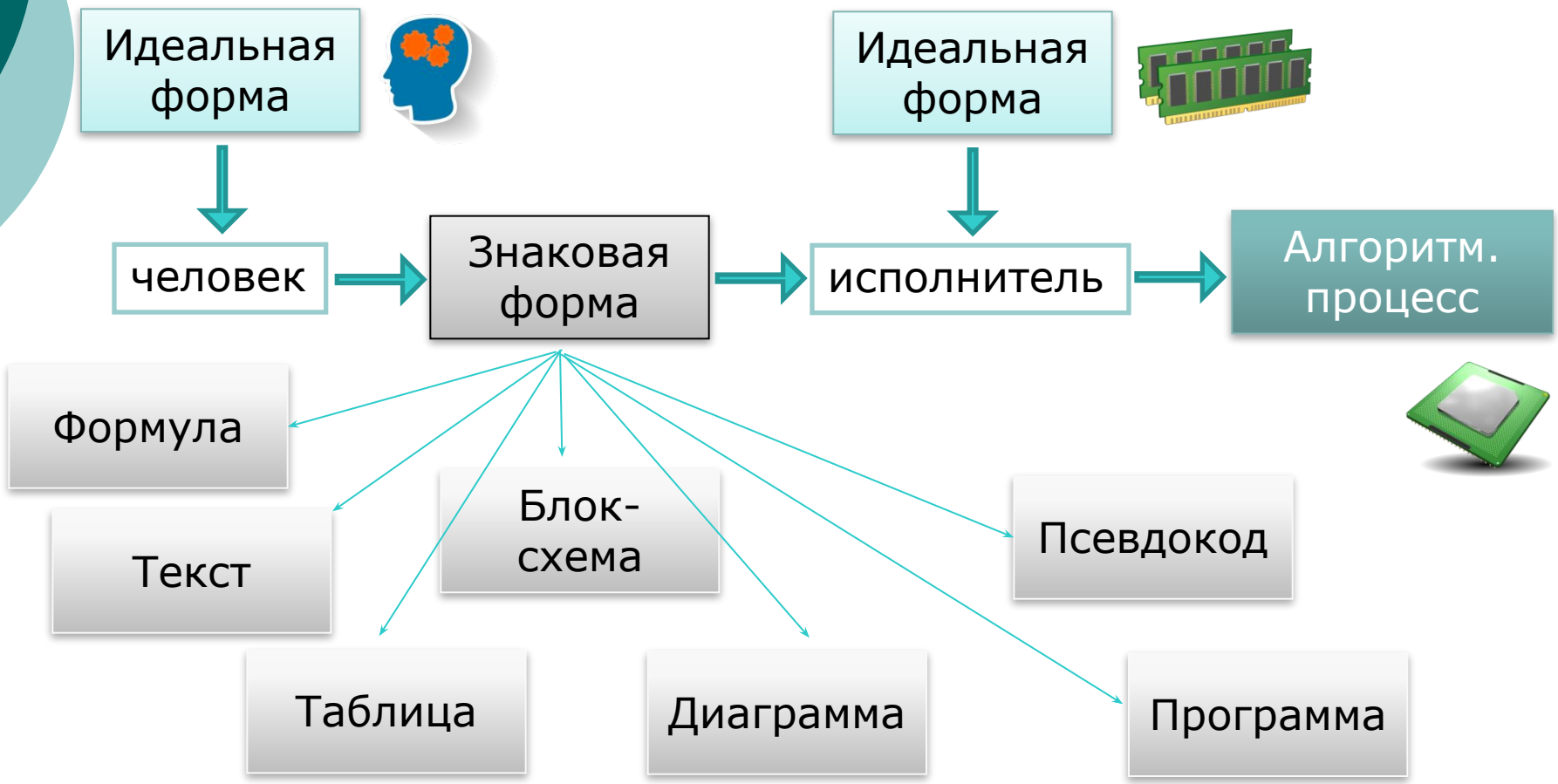
- Для учебного исполнителя **Землемер** составили алгоритм по расчету площади прямоугольного поля шириной А и длиной В:

Ввод А,В
Площадь = 2*(А+В)
Вывод Площадь

Логическая

Какой тип ошибки допущен в алгоритме ?

Формы представления алгоритма



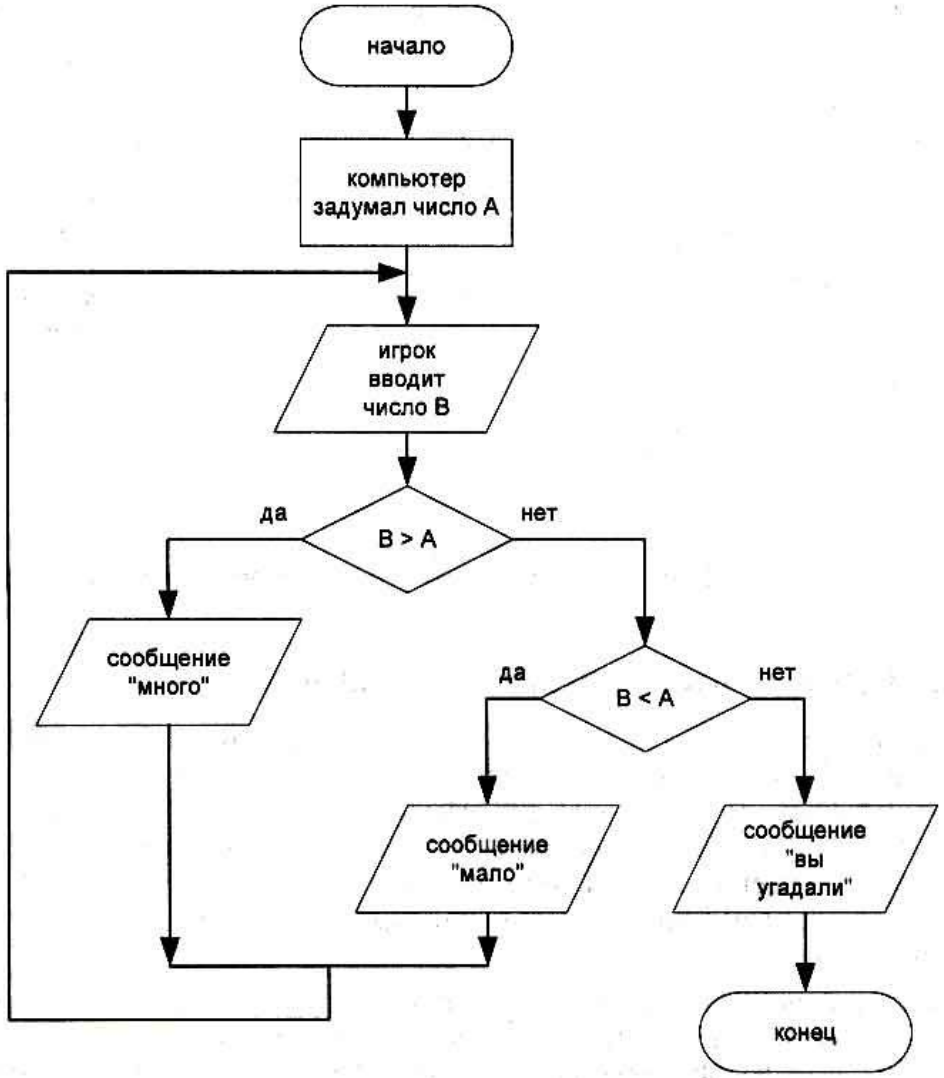
Формы представления алгоритма - текст

- задать два числа;
- если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
- определить большее из чисел;
- заменить большее из чисел разностью большего и меньшего из чисел;
- повторить алгоритм с шага 2.

Формы представления алгоритма - таблица

Свет в соседней комнате горит	**Да**	**Нет**	**Нет**
Свет у соседей горит	**-**	**Да**	**Нет**
Поменять лампочку	X		
Проверить пробки		X	
Позвонить электрику		X	X
Позвонить диспетчеру			X

Формы представления алгоритма – блок-схема



Формы представления алгоритма – диаграмма переходов

Формы представления алгоритма - псевдокод

```
алг сумма_квадратов
    дано | n>0
    надо | s = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i,n,s
    ввод n
    s:=0
    нц для i от 1 до n
        s:=s+i*i
    кц
    вывод “s = ", s
кон
```

Формы представления алгоритма - программа

```
program summ_kvadr;
//n>0
//S = 1*1 + 2*2 + 3*3 + ... + n*n
var i,n,s: integer;
begin
    readln (n);
    s:=0;
    for i:=1 to n do s:=s+i*i;
    writeln ('S = ', S);
end.
```

Формы представления алгоритма - программа

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD
10 E2 F9 CD 20 48 65 6C 6C 6F 2C 20
57 6F 72 6C 64 21

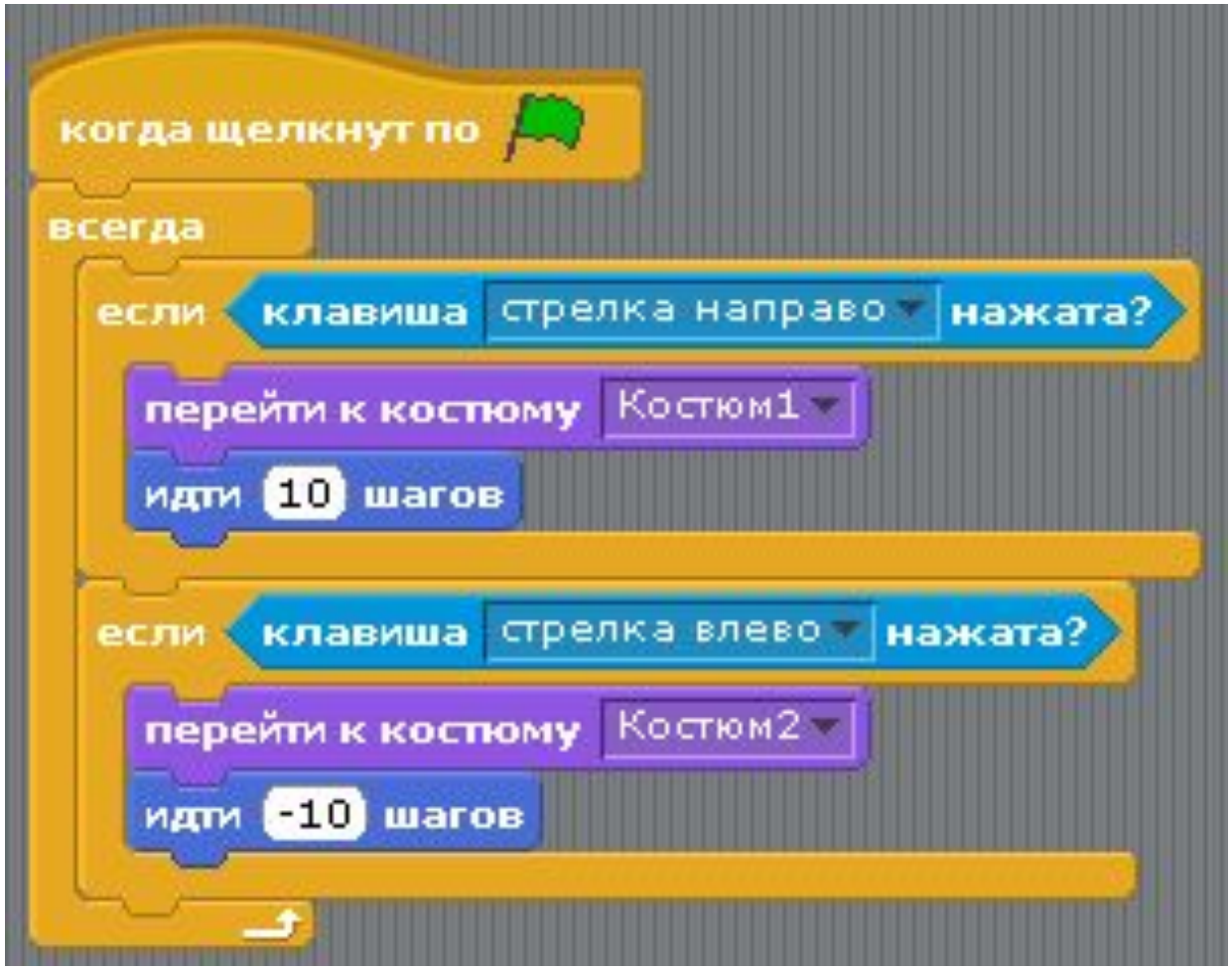
Важность графических форм

- Применение графическо-символьных представлений алгоритмов способствует обучению детей, у которых **наглядно-образный компонент** преобладает над **аналитическим**.
- Привлечение учебных исполнителей с **графическими обстановками** сориентировано на повышение наглядности и доступности алгоритмизации.

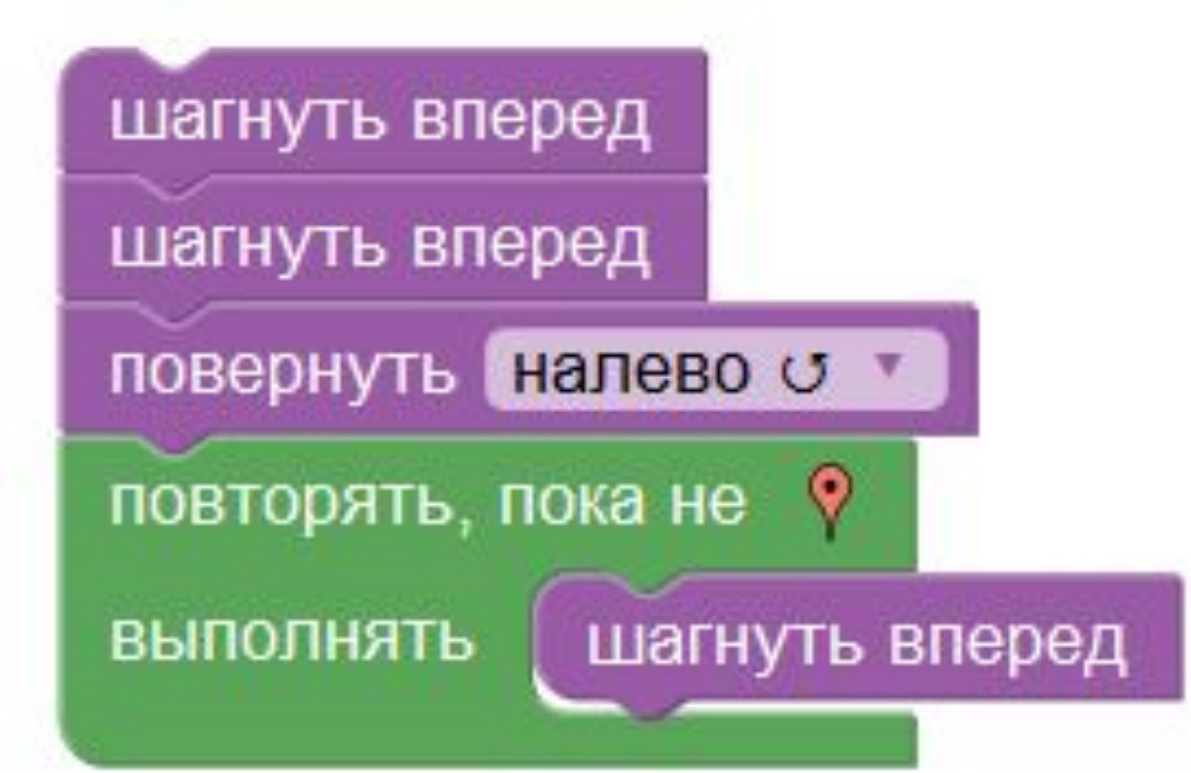
Важность графических форм

- Т.н. **«инженерное программирование»** также предполагает активное применение графических блоков при составлении алгоритма.
- Именно поэтому все чаще можно встретить такую форму представления, как **«Blockly»**, причем как для обучения, и для решения прикладных задач.

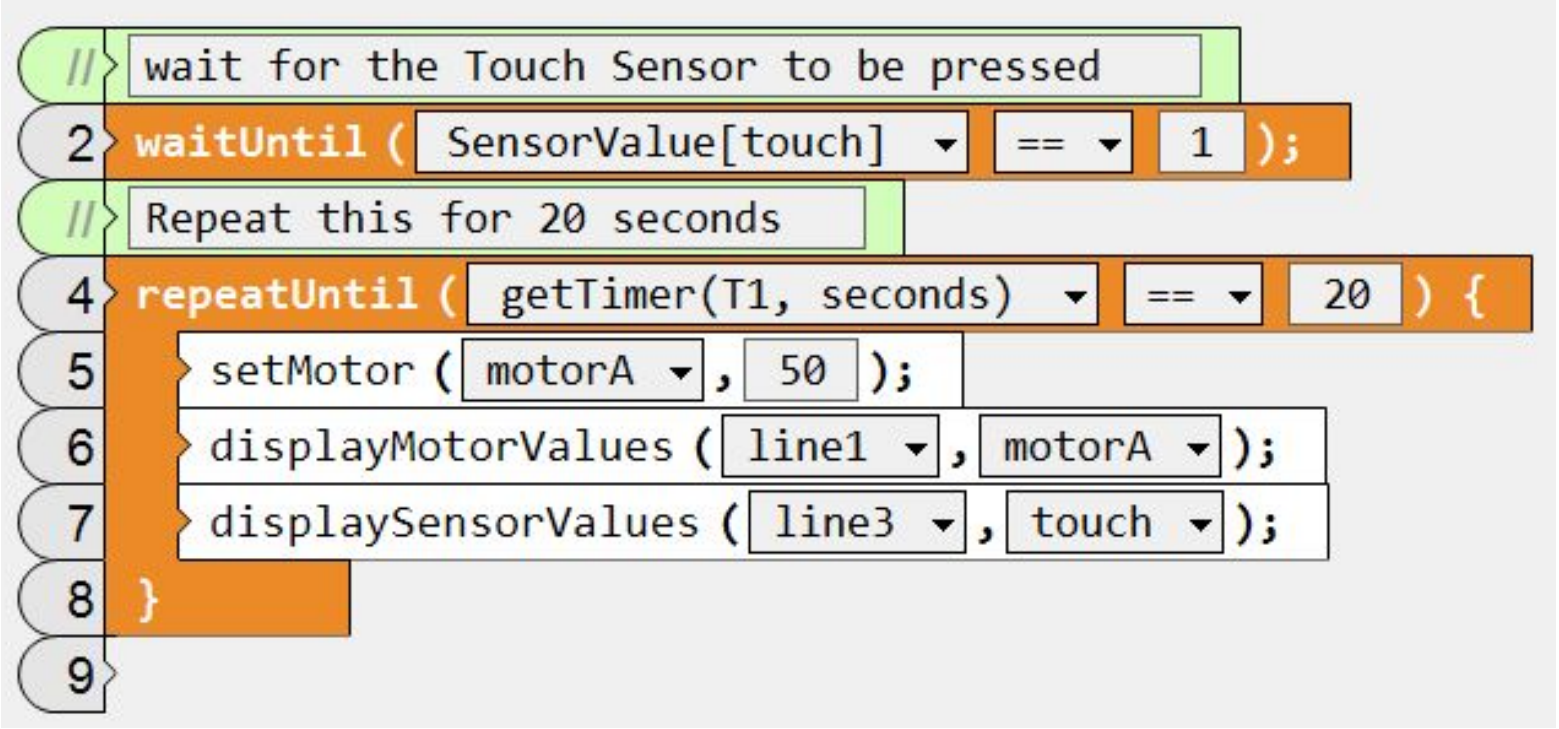
Blockly (Scratch)



Blockly (BlocklyGames)

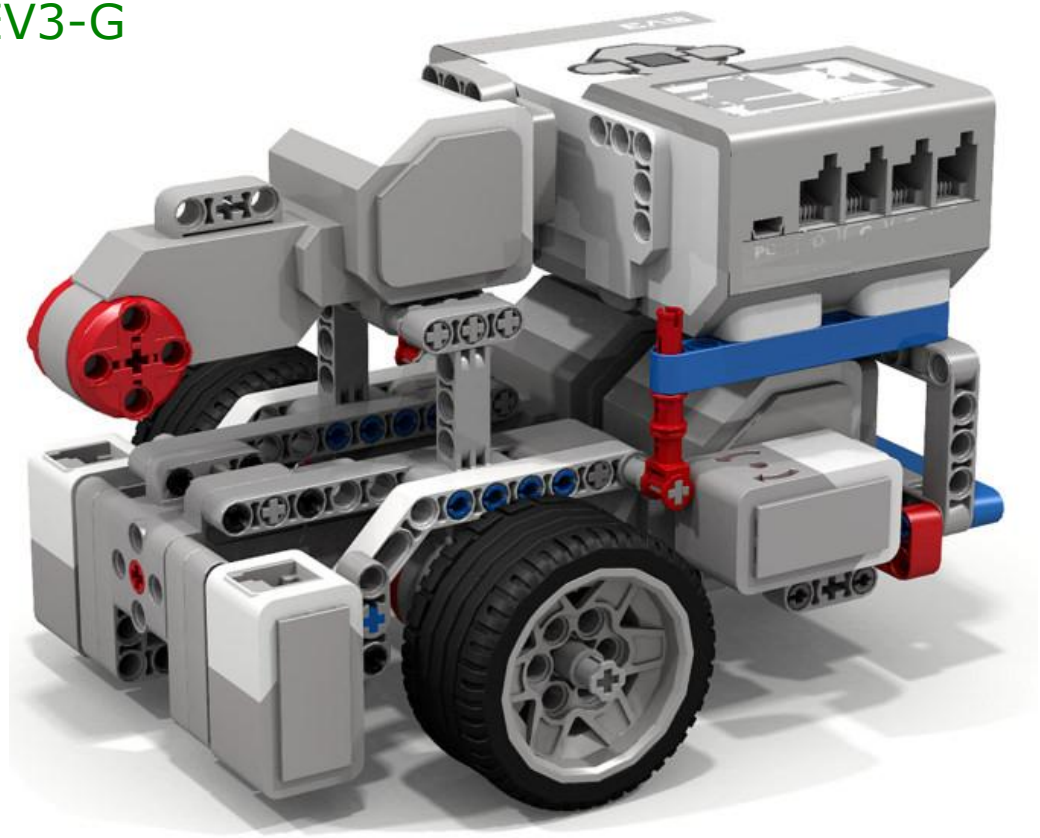


Blockly (RobotC)



Важность различных форм представления алгоритма/программы (Lego-EV3)

EV3-G



RobotC

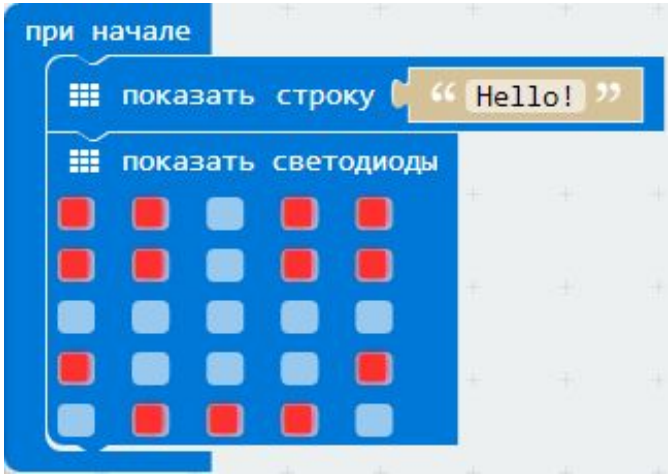
```
task main()
{
setMotor(motorB, 75);
setMotor(motorC, 75);
}
```

RoboBasic

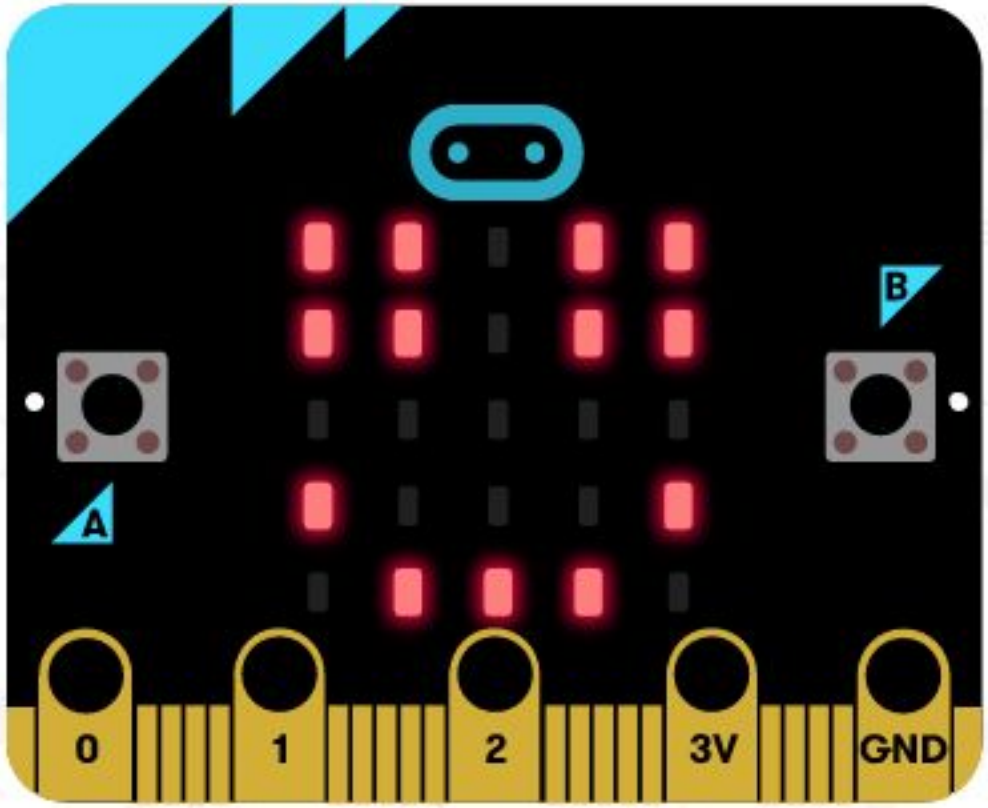
```
Motor.Move ("BC", 75, 360,
"True")
```

Порой текстовая запись короче графических форм

Важность различных форм представления алгоритма/программы (Microbit)




```
1  basic.showString("Hello!")
2  basic.showLeds(`
3      # # . # #
4      # # . # #
5      . . . . .
6      # . . . #
7      . # # # .
8      `)
```



Удобно для «начинающих» и «продолжающих»

Спасибо за внимание!

Обратная связь
http://nipkipro.ru/

Контакты
valovam@mail.ru

Федеральный Государственный Образовательный СТАНДАРТ