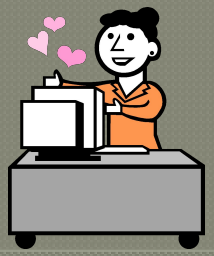
# Графический исполнитель

# ГРИС

## Инструмент для перехода к программированию

Обучающая презентация
для учеников 9-х классов.

# информатики ГБОУ СОШ № 515

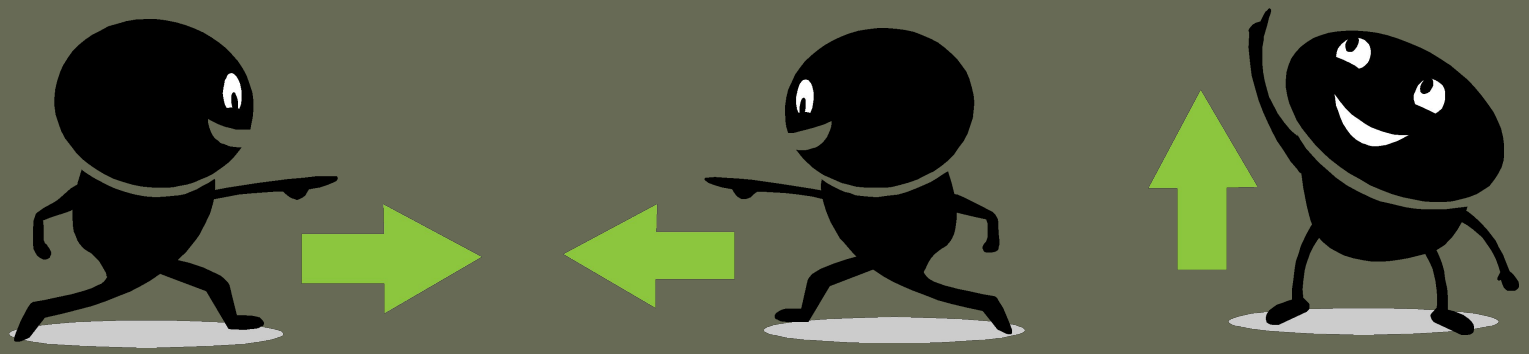


Юлия Владимировна Бабыкина

Если Вы закончили тему Алгоритмы, а впереди у вас программирование, данная презентация поможет вам адаптироваться и сделать данный переход более мягким.

# Управление

- Управление - важная часть нашей жизни. Видим мы это или нет, но успех любого предприятия зависит от того, насколько четко даны инструкции, насколько они понятны. Кроме того, они должны учитывать знания и способности того, кто эти инструкции будет выполнять.



Если мы говорим об общении человека с компьютером, нужно понимать, что компьютер - машина. Он может считать с огромной скоростью, совершать многие операции одновременно.

---

Но он совершенно не умеет принимать самостоятельные решения, даже если они логичны и очевидны.

# СКИ

- Если мы хотим, что бы компьютер исполнял наши приказы, нужно уметь объяснять чего мы хотим, на понятном ему я[illegible].

- Программа - это алгоритм написанный на языке исполнителя.
- Язык исполнителя, он же система команд исполнителя (СКИ) - это его умения.

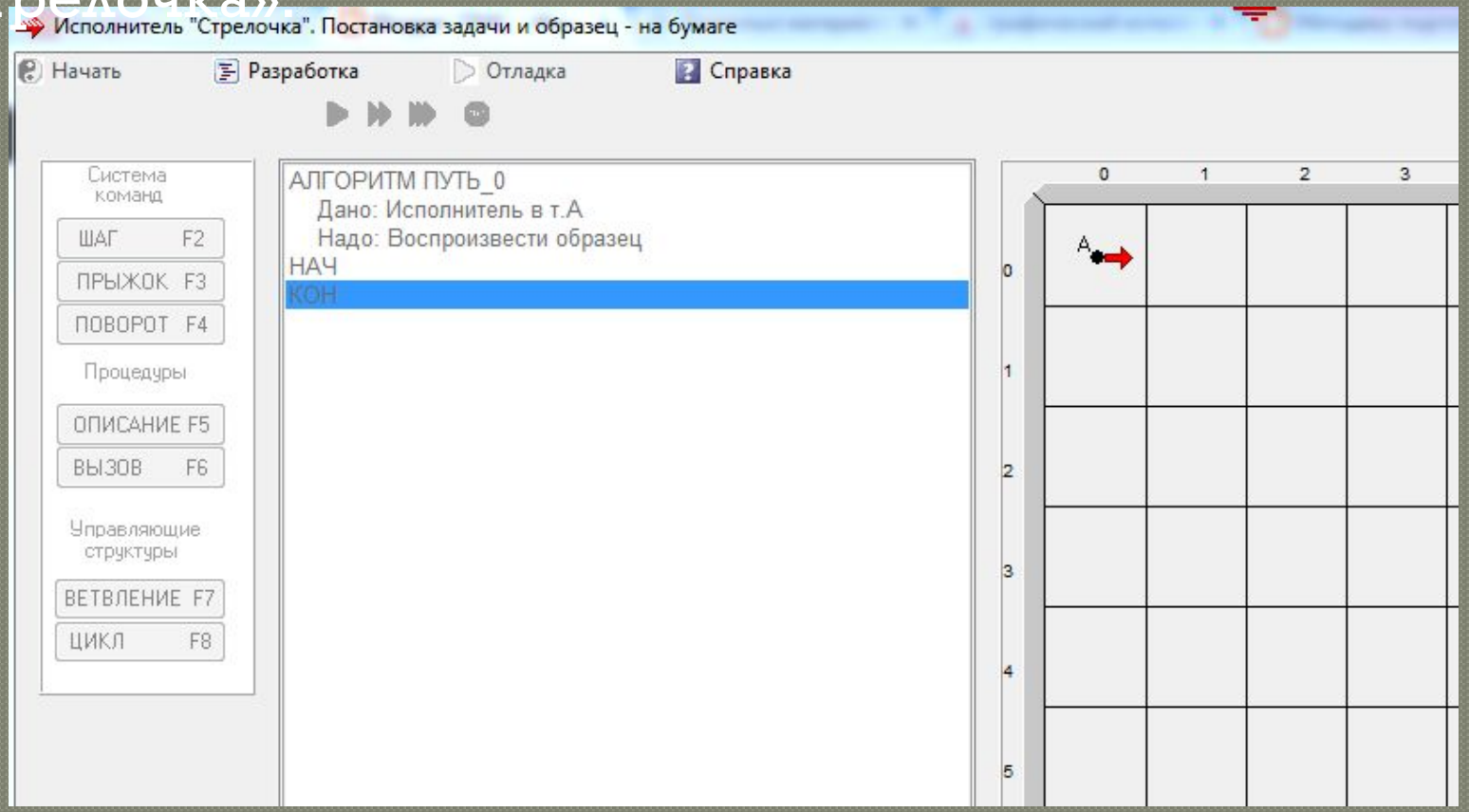
# «Стрелочка»

программа разработанная для того, что бы наглядно демонстрировать процесс управления.

- Вот простые команды, которые умеет делать ГРИС «Стрелочка»:
  **шаг** - перемещение ГРИС на одну клетку вперед с рисованием линии;
  **поворот** - поворот направления движения на 90° против часовой стрелки;
  **прыжок** - перемещение вперед на одну клетку, без рисования линии.

# Обстановка, в которой действует исполнитель, называется средой исполнителя.

Посмотрите, как выглядит среда исполнителя «Стрелочка»:



# Тело программы

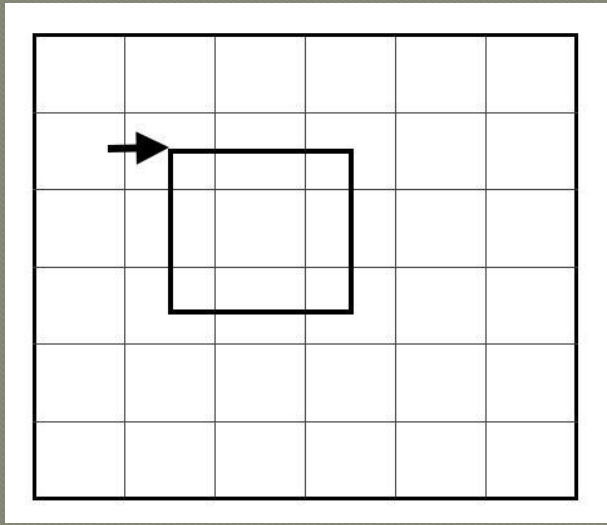
- Тест программы, а правильнее сказать - тело программы, должно, так же как и в блок-схеме, размещаться между словами Начало и Конец.

В ГРИС используют сокращение:
НАЧ и КОН.

- Все команды написанные выше или ниже, не будут учтены программой, а значит, они не будут выполнены.

# Попробуем решить задачу:

Построим квадрат со стороной 2 клетки. Исходное положение ГРИС примем за левый верхний угол квадрата. Стрелка направлена вправо.

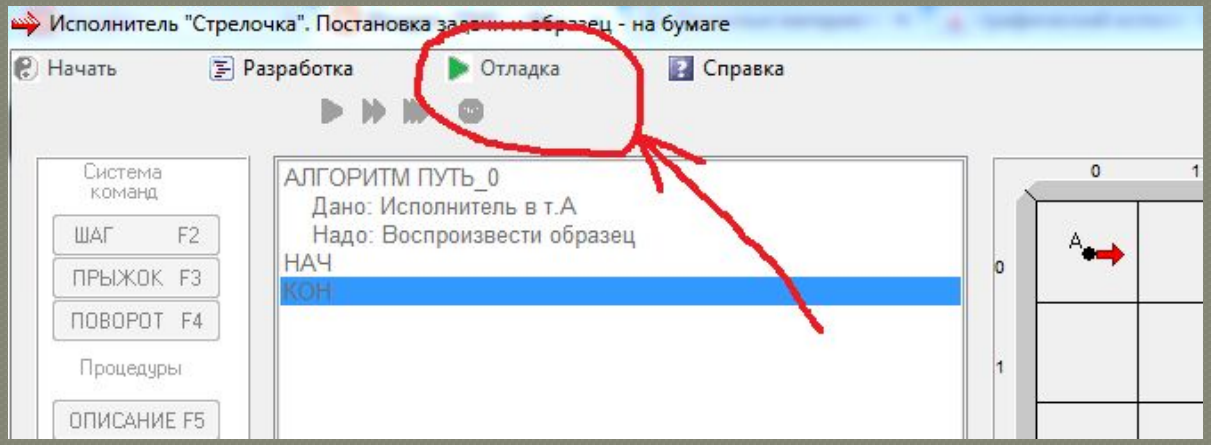


Тест программы:

ШАГ ШАГ ПОВОРОТ ПОВОРОТ ПОВОРОТ

ШАГ ШАГ ПОВОРОТ ПОВОРОТ ПОВОРОТ

ШАГ ШАГ ПОВОРОТ ПОВОРОТ ПОВОРОТ

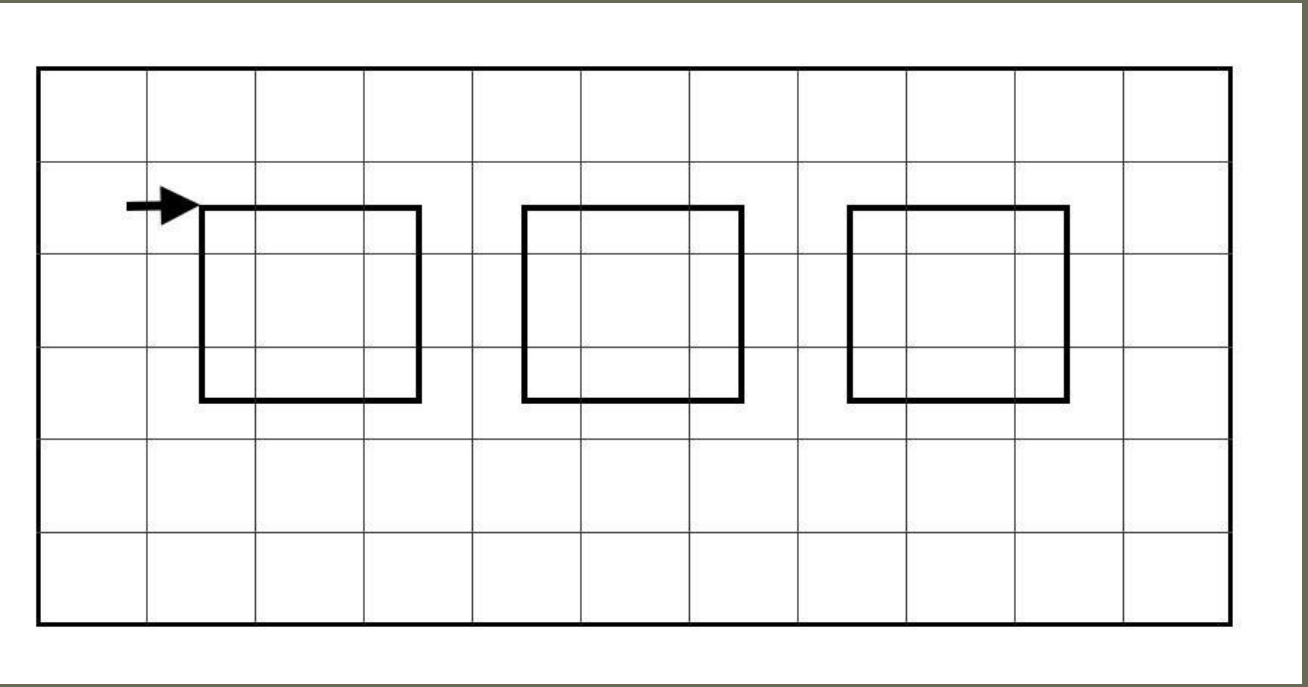



Теперь найдите кнопку с изображением зеленой стрелки - это отладка. Данный режим позволяет нам просмотреть как ГРИС исполняет написанную программу.

Вы можете выбрать скорость просмотра.

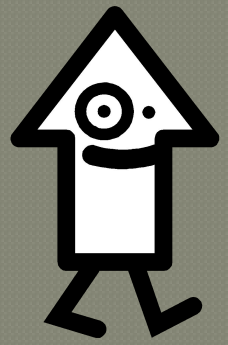
# Задача № 2:

Теперь давайте построим три квадрата со стороной два шага. Разместить квадраты нужно в одну горизонтальную линию. Расстояние между квадратами - один шаг.

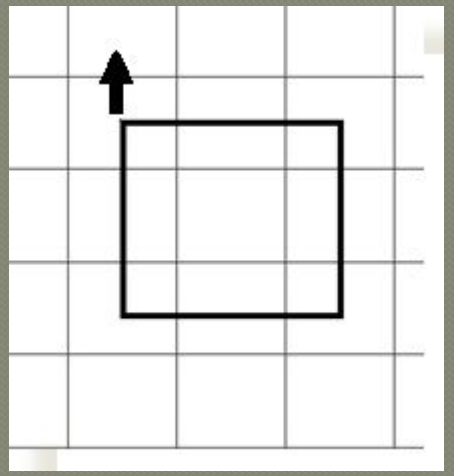


# Решение:

Несколько раз повторяется одно и тоже действие. Но мы не можем просто скопировать текст предыдущей программы и повторить его три раза.



Дело в том, что закончив рисование первого квадрата ГРИС остановится так, что стрелка будет стоять в начальной точке и смотреть вверх.



# Вспомогательный алгоритм

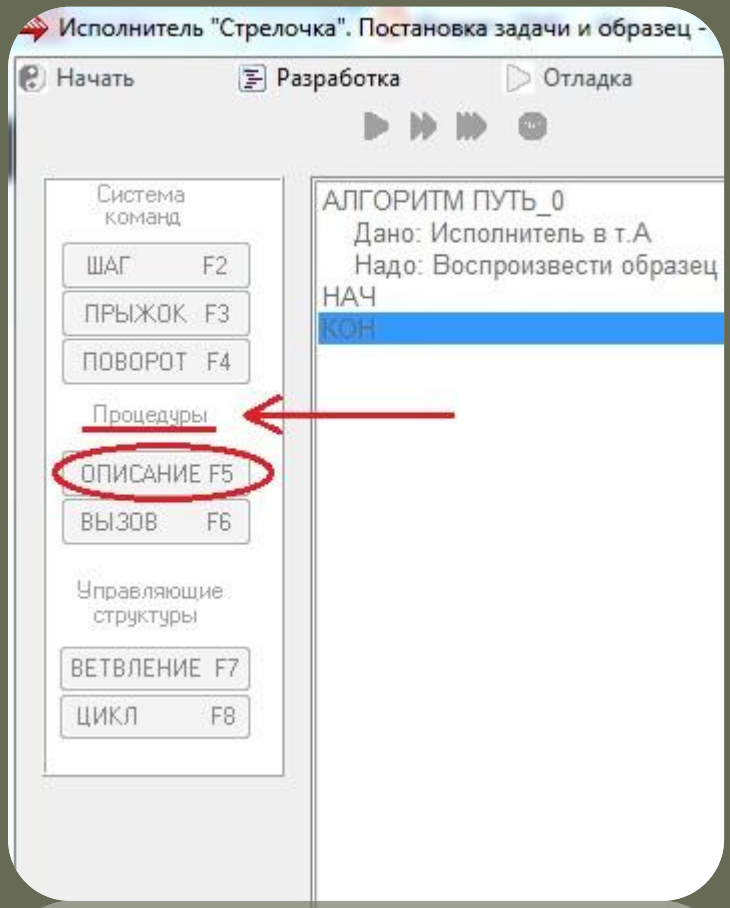
- Если в алгоритме повторяются какие-либо действия - используют вспомогательные алгоритмы. Это такие алгоритмы, которые решают некую подзадачу из основной задачи и повторяются многократно.

В нашем примере, это может быть самостоятельный алгоритм - построение одного квадрата (1-я задача).

В языке программирования такие алгоритмы называют подпрограммами или процедурами.

# Процедура

- Для построения процедуры Квадрат, нужно в левой части экрана выбрать кнопку - Описание процедуры. У вас появляется окно, в котором вы должны задать имя новой процедуры, т.е. для нас - Квадрат.



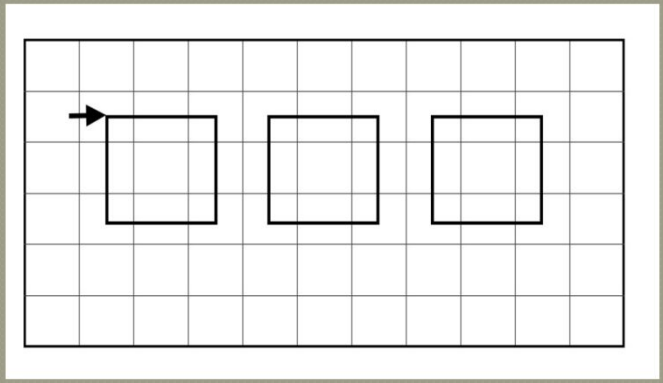
# Текст процедуры

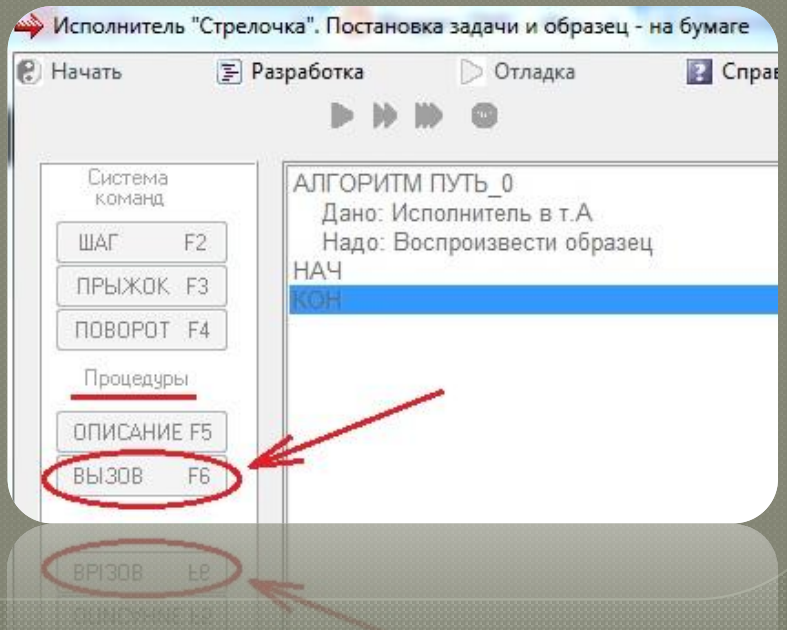
- Закройте окно и увидите, что слева появилось начало и конец процедуры - между ними вы пишете текст процедуры:

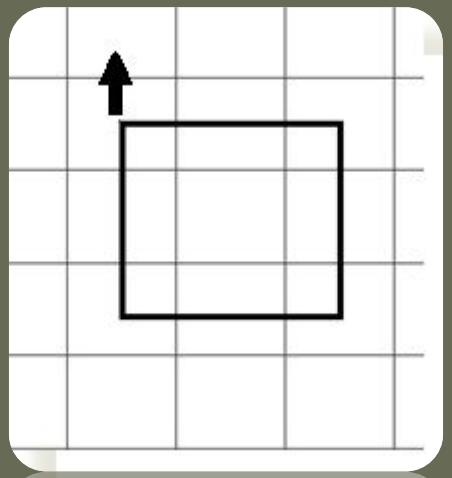
```
ШАГ  ШАГ
ПОВОРОТ
ПОВОРОТ  ПОВОРОТ
ШАГ  ШАГ
ПОВОРОТ  ПОВОРОТ  ПОВОРОТ  ШАГ
ШАГ
ПОВОРОТ  ПОВОРОТ  ПОВОРОТ
ШАГ  ШАГ
```

# Основной алгоритм

- Теперь нужно построить основной алгоритм Строка квадратов.
- Сначала мы вызываем процедуру Квадрат, соответствующей кнопкой в левой части экрана.

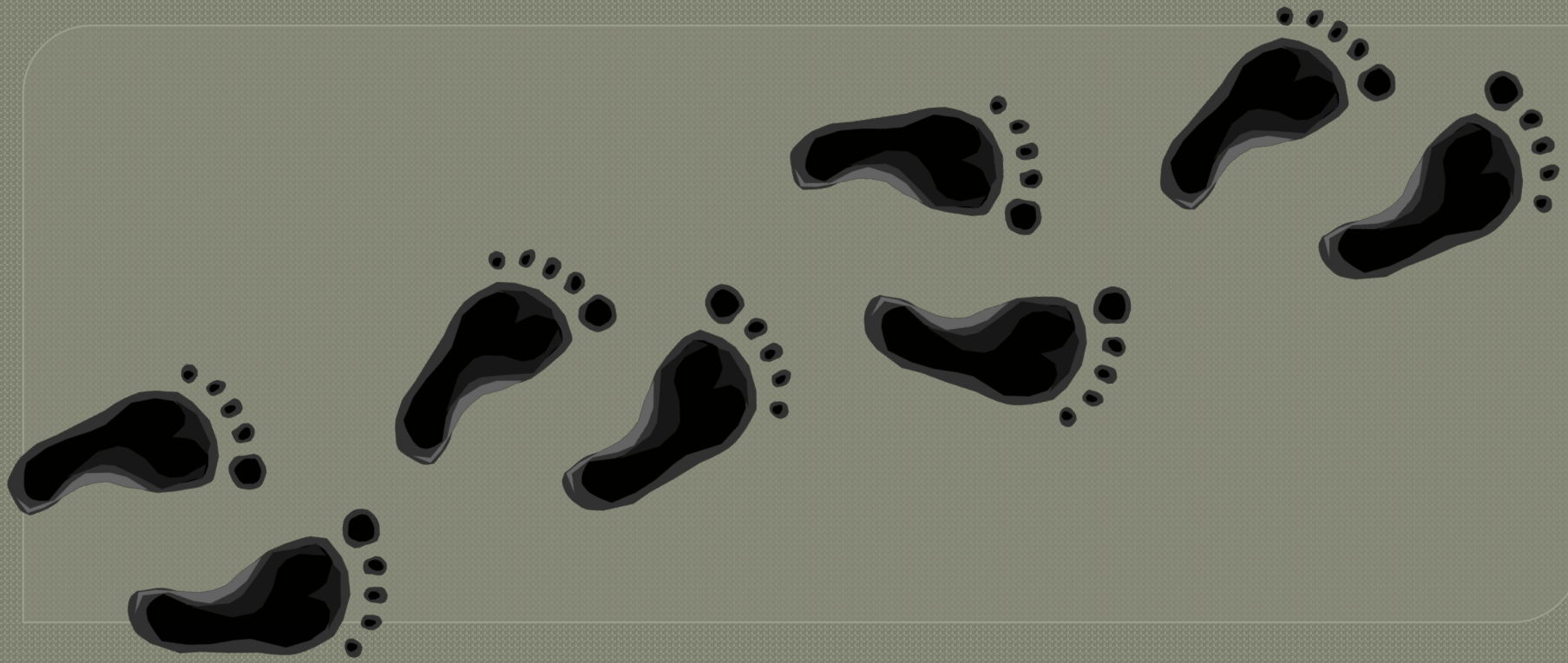






# Между процедурами

После выполнения процедуры Квадрат, ГРИС стоит в начальном положении, но стрелка смотрит вверх. Значит требуется написать шаги, приводящие ее в необходимое для нас положение - когда стрелка стоит в левом верхнем углу будущего квадрата и смотрит вправо.



Необходимые шаги:

ПОВОРОТ ПОВОРОТ ПОВОРОТ

ПРЫЖОК ПРЫЖОК ПРЫЖОК

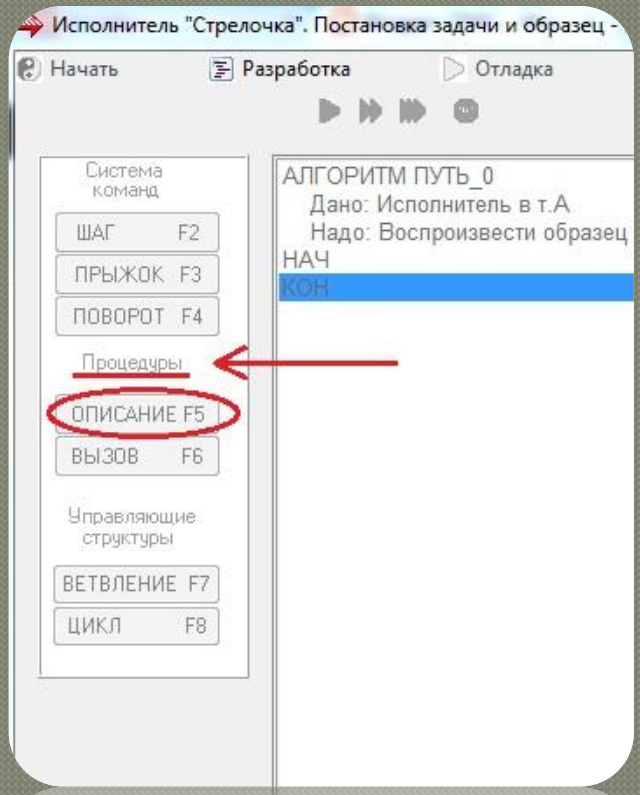
## Программа:

```
сделай Квадрат
ПОВОРОТ  ПОВОРОТ  ПОВОРОТ
ПРЫЖОК  ПРЫЖОК  ПРЫЖОК
сделай Квадрат
ПОВОРОТ  ПОВОРОТ  ПОВОРОТ
ПРЫЖОК  ПРЫЖОК  ПРЫЖОК
сделай Квадрат
```

Это полный текст программы, необходимой для рисования строки из 3-х квадратов.

Но в программе может быть и несколько процедур. У нас повторяется текст программы, отвечающий за разворот стрелки в нужное направление.

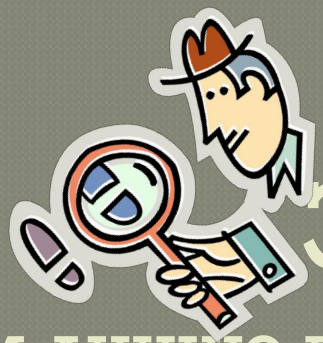
Можно создать новую процедуру – Разворот. Тогда текст программы будет гораздо короче.



# Окончательный текст программы

С добавлением новой процедуры, текст программы будет таким:

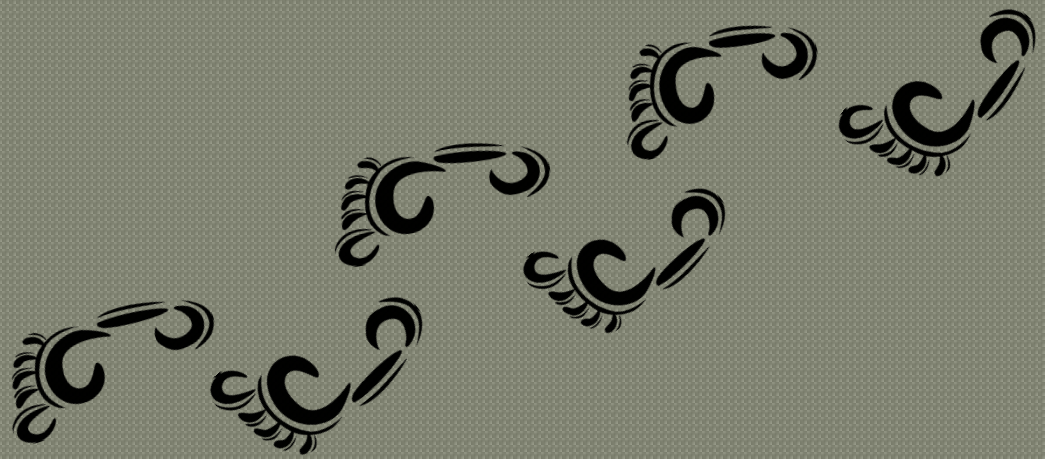
- **сделай** Квадрат
- **сделай** Разворот
- **сделай** Квадрат
- **сделай** Разворот
- **сделай** Квадрат



# Задача 3.

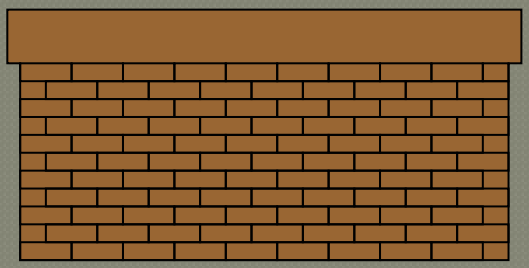
Давайте построим линию на всю ширину поля.

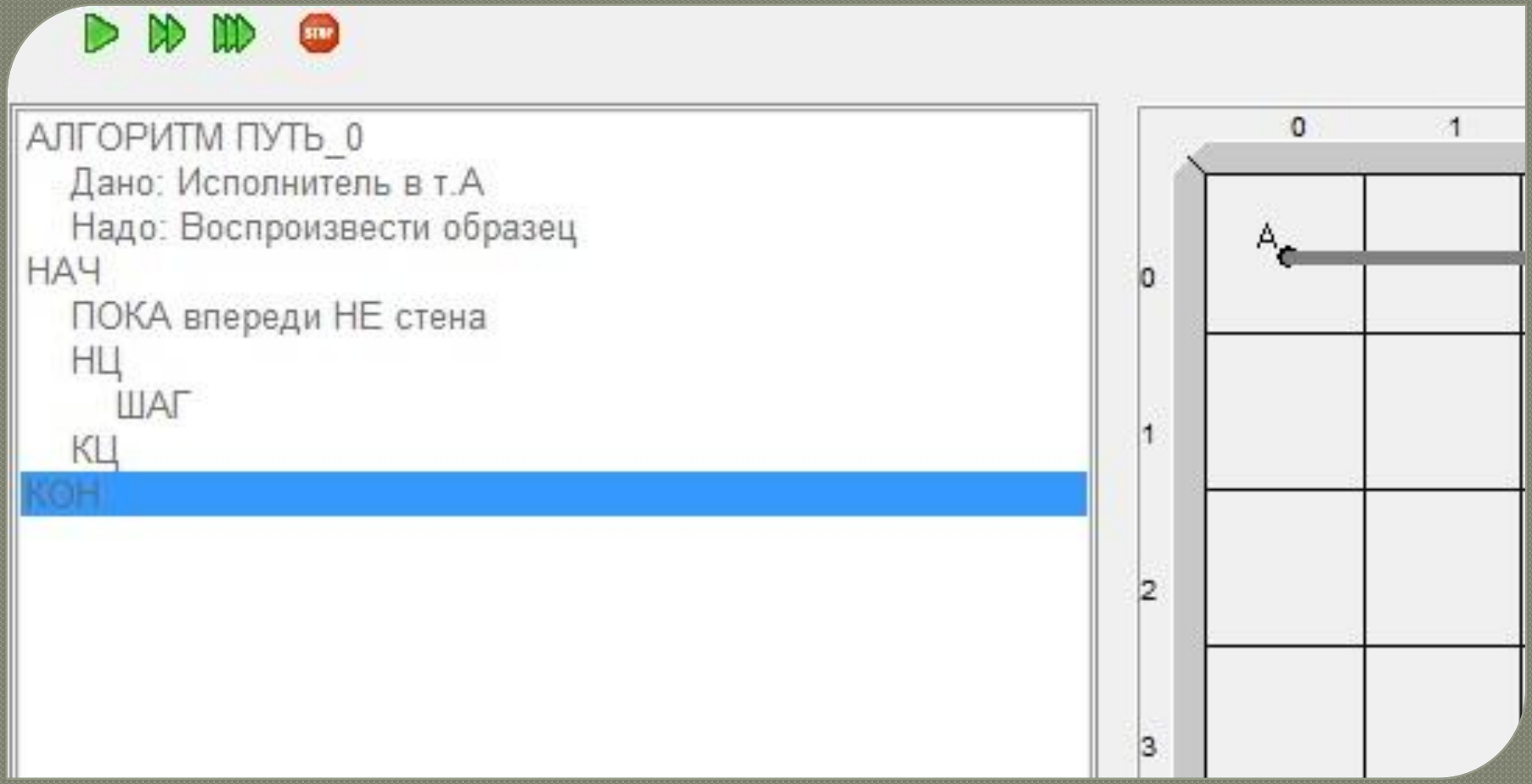
Можно сосчитать количество клеток и столько раз нажать кнопку ШАГ.

# Правильное решение

Но для подобных задач, у ГРИС существует специальная команда.
Как вы понимаете, это цикл. Циклический алгоритм - это алгоритм в котором команда или группа команд выполняется многократно, пока не выполнено условие.
В нашем случае условие: делать шаг, пока не дойдем до края поля.
В ГРИС край поля так же называется стеной.






**Мы выбираем условие:**
**пока впереди НЕ стена**

- В одной программе можно использовать не только несколько процедур, но и несколько циклов.

*Задача 4.*
*Постройте рамку по периметру рабочего поля ГРИС.*

# Решение.

Назовем алгоритм Рамка и используем все полученные знания, что бы сделать программу максимально лаконичной. ГРИС умеет поворачиваться только против часовой стрелки, поэтому, что бы не делать много лишних поворотов, удобнее повернуть стрелку вниз, тогда разворот в каждом углу рамки мы сможем делать используя только одну команду ПОВОРОТ.

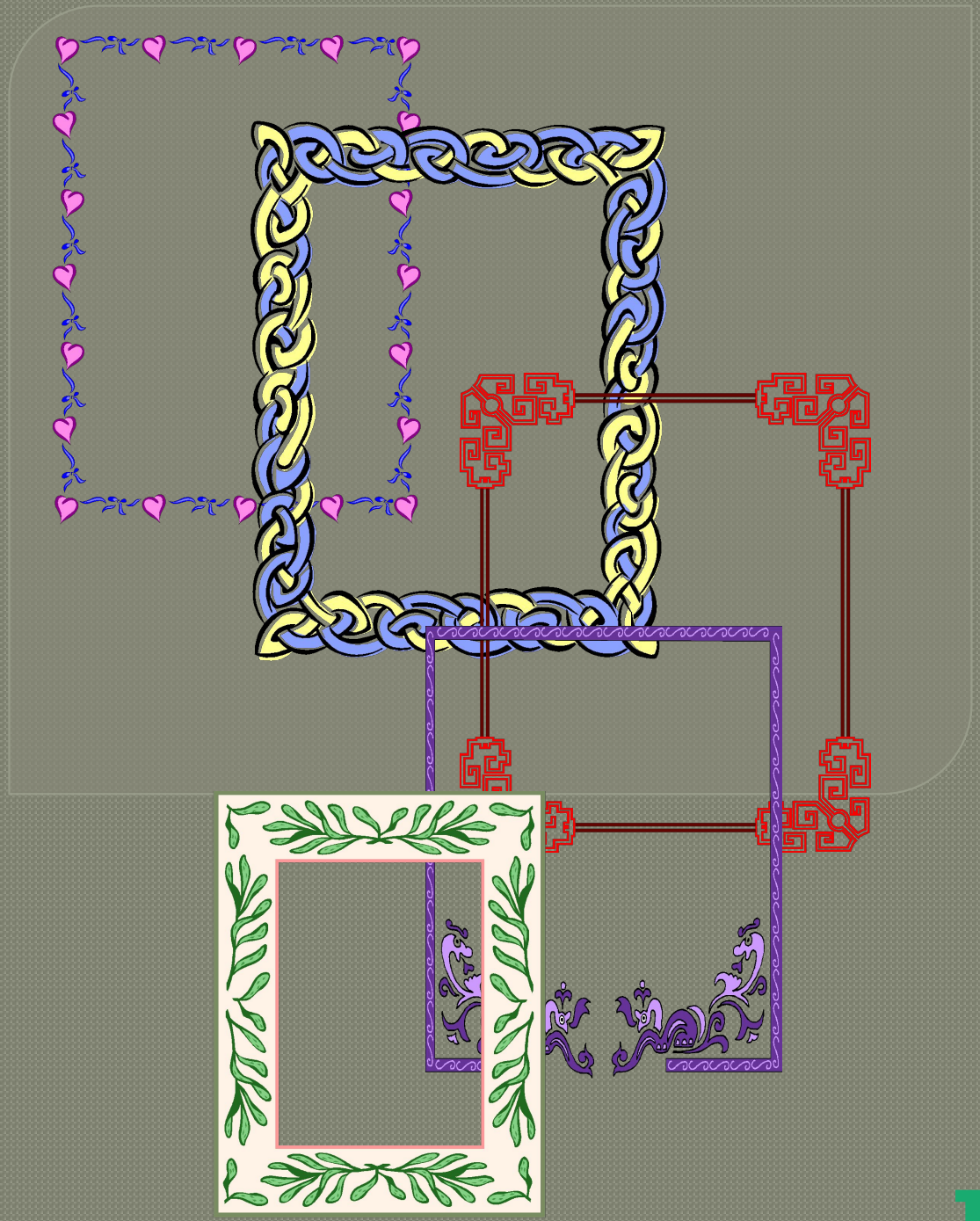
Создадим процедуру линия, как делали это в прошлой задаче:

**процедура ЛИНИЯ**

**пока впереди не край, повторять**

ШАГ

**конец процедуры**



**Основной алгоритм Рамка:**

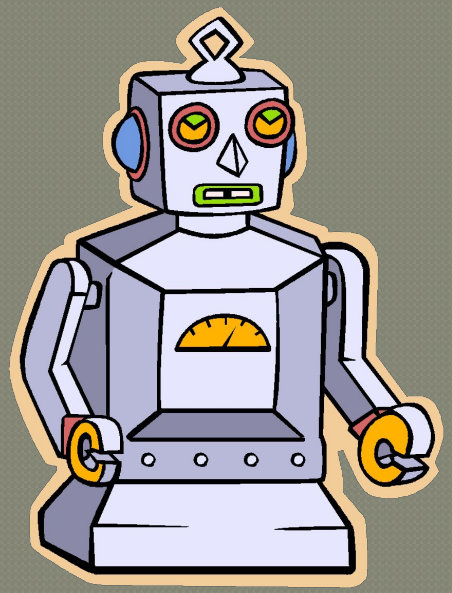
**ПОВОРОТ**
**ПОВОРОТ**
**ПОВОРОТ**
**сделай ЛИНИЯ**
**ПОВОРОТ**
**сделай ЛИНИЯ**
**ПОВОРОТ**
**сделай ЛИНИЯ**
**ПОВОРОТ**
**сделай ЛИНИЯ**
**конец цикла**

**РАМКА ГОТОВА!**

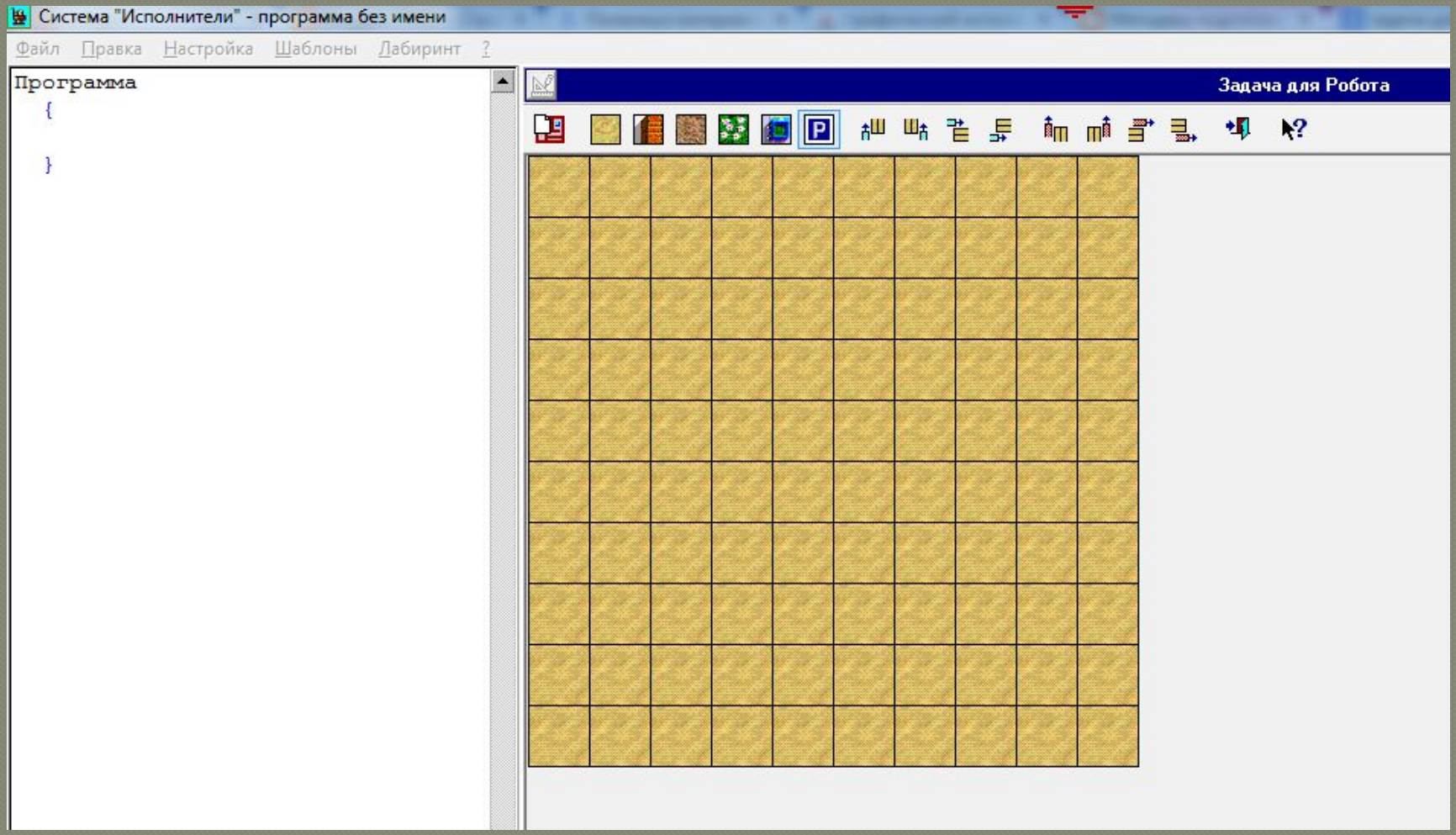
# Теперь вы умеете управляться с ГРИС «Стрелочка».

- Можно сказать – это первый шаг к программировани ю.
- Второй шаг – более сложный ГРИС – тот у которого больше умен (СКИ)

- Я предлагаю вам познакомитьсяя с ГРИС – РОБОТ.



# Перед вами среда исполнителя Робот:



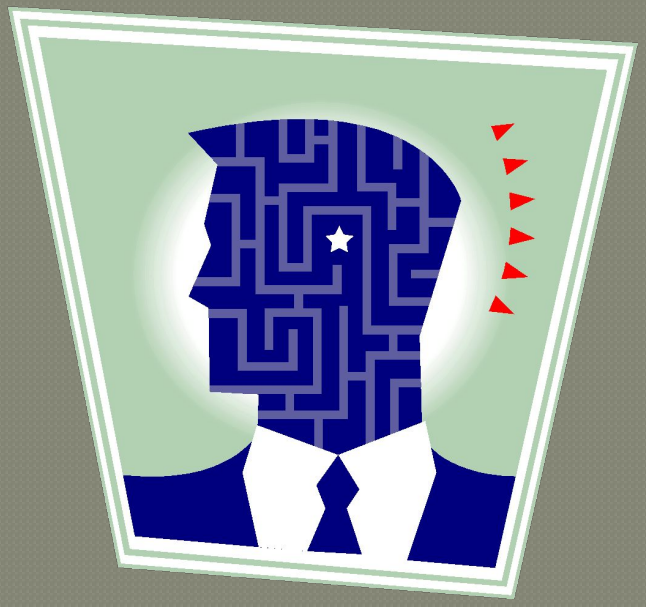
# Подготовка к работе

Для начала работы, нам нужен лабиринт, который будет проходить наш Робот.
И его вы создадите сами. В дальнейшем это можно использовать не только как тренировку, но и как развлечение.

# Инструменты

- Зайдите во вкладку Лабиринт и выберите - Редактировать лабиринт.
- Первый значок - Создать новый лабиринт

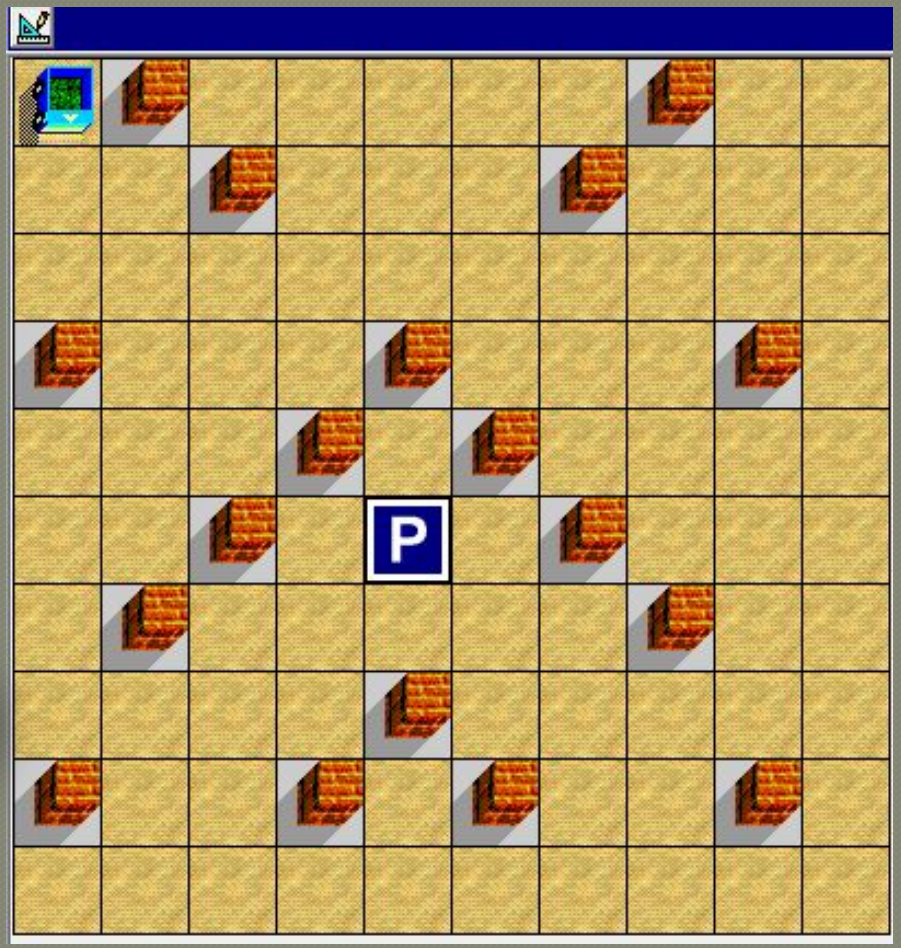
ДАЛЕЕ ОБЪЕКТЫ: ПЕСОК (ОСНОВНОЙ ФОН), СТЕНА И ЦВЕТЫ (ПРЕГРАДЫ), ГРЯДКА, ПРЕДПОСЛЕДНЯЯ - САМ РОБОТ. А ПОСЛЕДНИЙ - БАЗА. КЛЕТКА, ГДЕ ВЫ РАЗМЕСТИТЕ РОБОТА СТАРТ, БАЗА - ФИНИШ (ИМЕННО В КЛЕТКУ БАЗЫ ДОЛЖЕН ПРИЙТИ РОБОТ).



**Нарисовав лабиринт, сохраните его себе на компьютер. ВАЖНО! Что бы он сохранился и в последующем работал, после придуманного вами имени лабиринта, необходимо поставить расширение .maz**

# Задача 1.

- Строим лабиринт и алгоритм действий, для перемещения робота на базу.



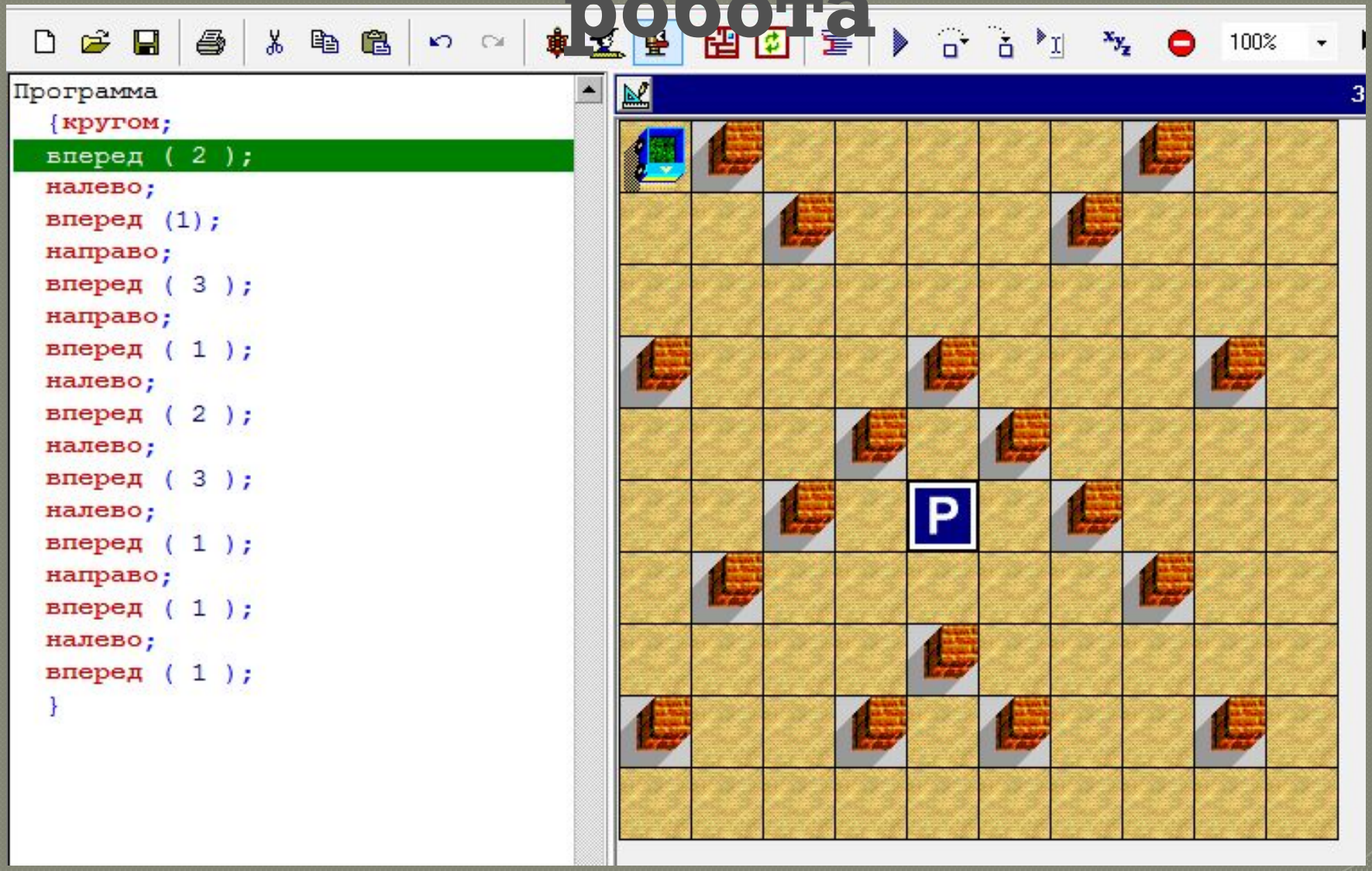
# СКИ Робота.

Вот алгоритм, состоящий из простых команд, входящих в СКИ Робота.

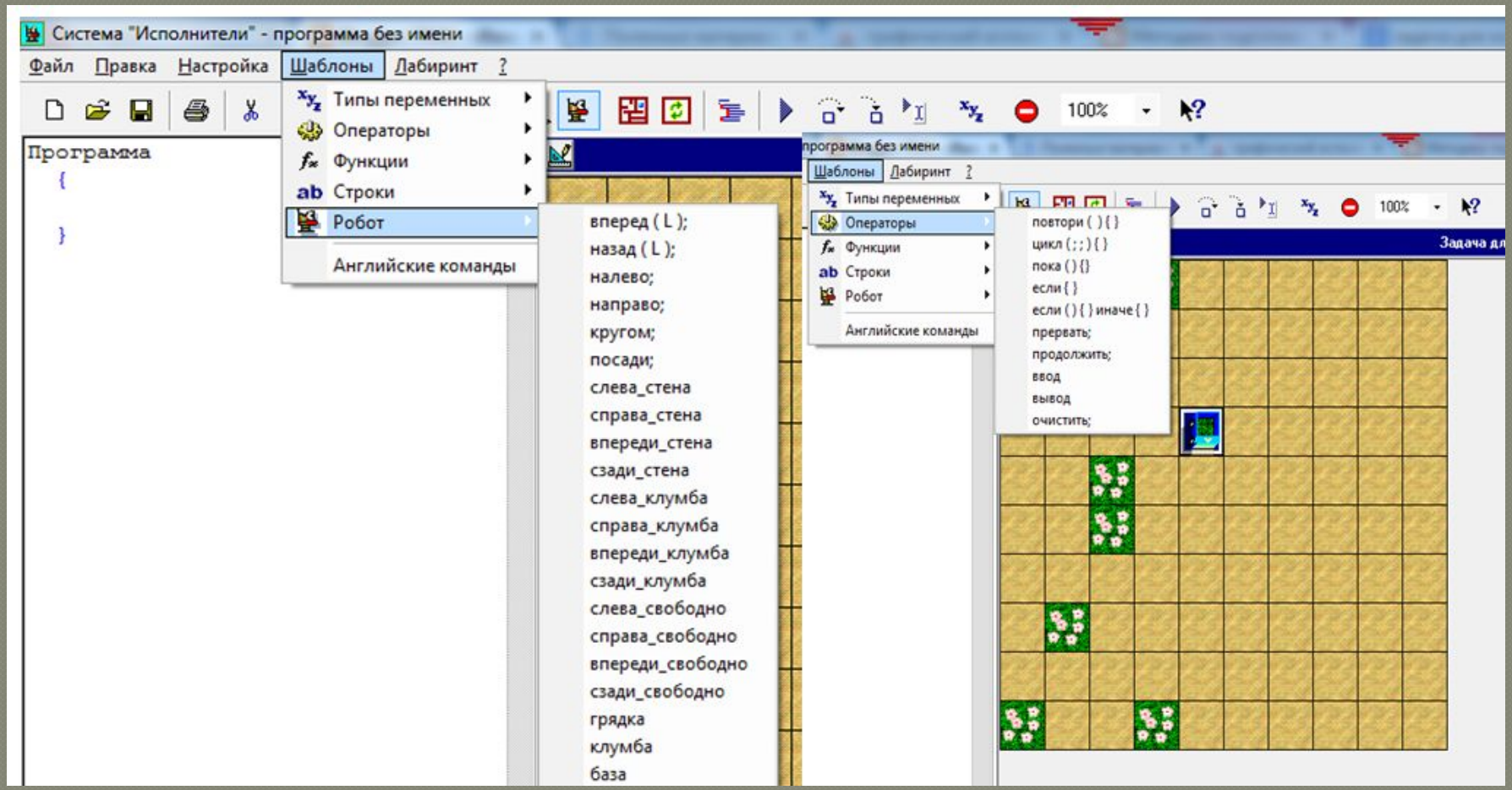
```
Программа
  {кругом;
  вперед ( 2 );
  налево;
  вперед (1);
  направо;
  вперед ( 3 );
  направо;
  вперед ( 1 );
  налево;
  вперед ( 2 );
  налево;
  вперед ( 3 );
  налево;
  вперед ( 1 );
  направо;
  вперед ( 1 );
  налево;
  вперед ( 1 );
  }
```

- Теперь мы задаем направление движению (вверх, вниз, направо, налево).
- Так же как и в языках программирования, в исполнителе Робот, после каждой команды ставится знак - ; Если его не поставить - исполнитель не выполнит команду.
- Если подряд идет повторение одной и той же команды, в скобках можно

# Проследите движения робота



# Открыв вкладку Шаблоны, изучите весь СКИ Робота!



# Задача 2.

Условия: Робот стоит в верхнем левом углу, направление движения - вниз. По диагонали, из левого угла расставлены грядки. Последняя клетка - нижний правый угол поля - база. Задача - написать алгоритм, по которому робот передвигается на базу, по ...ть высаживая на все грядки цветы.

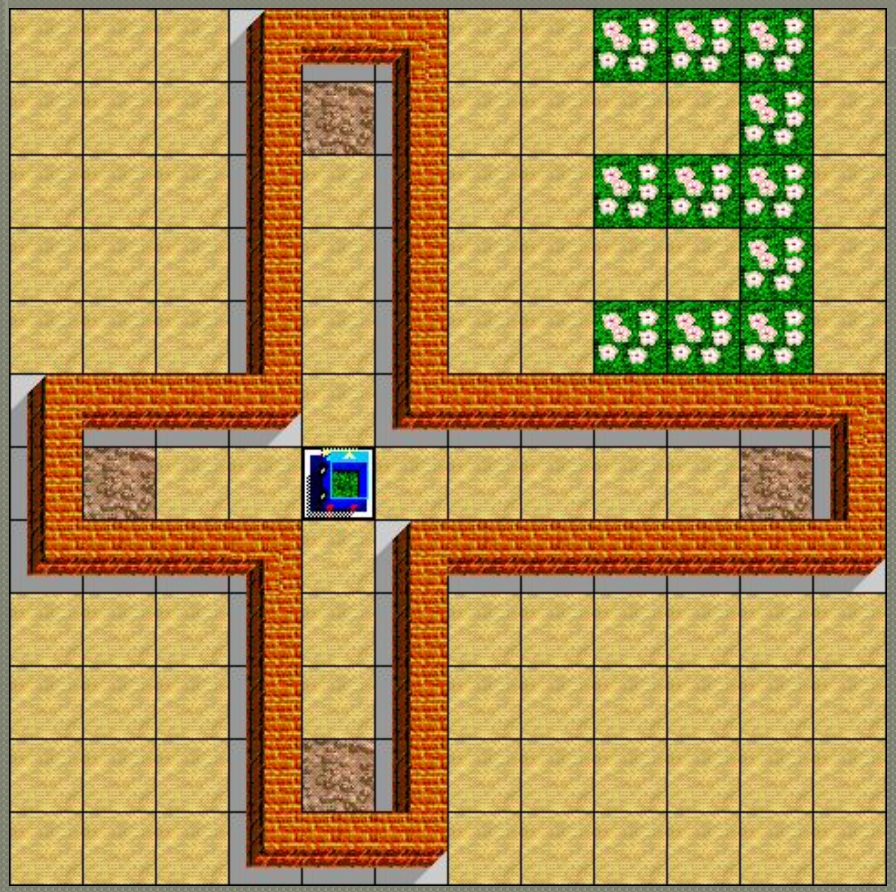
# Решение.

```
Программа
  {пока ( впереди_свободно)
  {вперед ( 1 );
  налево;
  вперед ( 1 );
  если ( грядка )
    { посади;}
  иначе
    { база;} направо;}

     }
```


Это пример написания цикла в ГРИС Робот.

# Используя свои знания и опыт, выполните задание.



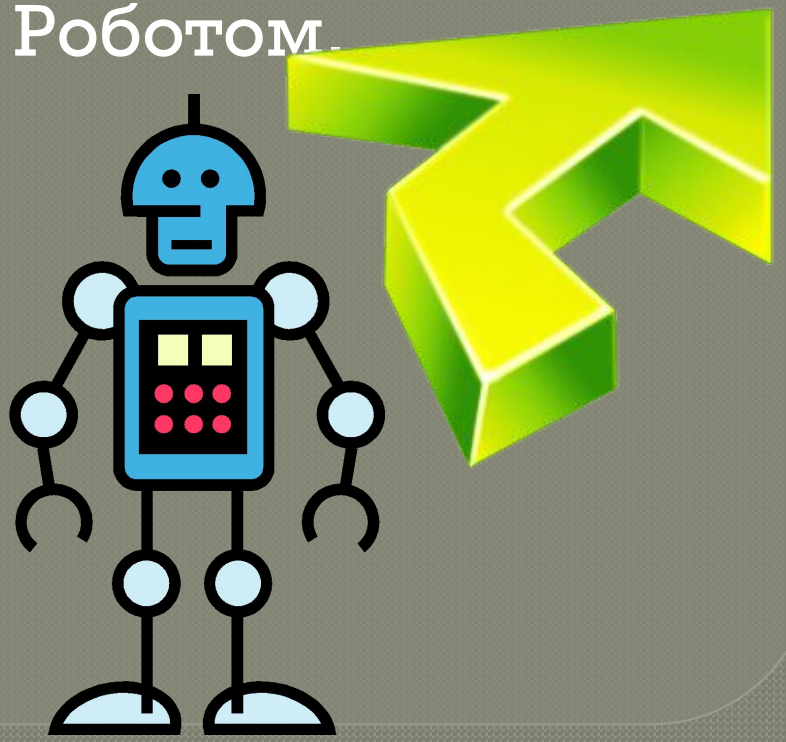
Постройте лабиринт в форме креста, робот и база находятся в центре. Исходное положение – робот смотрит вверх. Посадите цветы на всех свободных грядках внутри лабиринта.

# Можно придумать много интересных задач для робота.

Но нужно двигаться дальше.

- Новый этап подготовки к программированию – это программа Кумир.
- Она не является ГРИС и гораздо больше похожа на настоящую систему программирования.

- Но, вы наверняка увидите много общего со Стрелочкой и Роботом.



# У Кумира очень большой СКИ!
# А значит и возможности!

## КОМАНДЫ РОБОТА

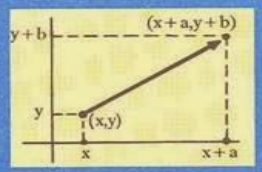
вверх вниз
вправо влево
закрасить

лог сверху стена — лог сверху свободно
лог снизу стена — лог снизу свободно
лог справа стена — лог справа свободно
лог слева стена — лог слева свободно
лог клетка закрашена — лог клетка не закрашена

вещ температура
вещ радиация

## КОМАНДЫ ЧЕРТЕЖНИКА

```
поднять перо
опустить перо
сместиться в точку( арг вещ x, y )
сместиться на вектор( арг вещ a, b )
```




## ОБЩИЙ ВИД АЛГОРИТМА

```
алг имя( аргументы и результаты )
 дано условия применимости алгоритма
 надо цель выполнения алгоритма
нач описание промежуточных величин
   тело алгоритма (последовательность команд)
кон
```

## КОМАНДЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА

```
нц число повторений раз
| тело цикла (последовательность команд)
кц
```

```
нц пока условие
| тело цикла (последовательность команд)
кц
```

```
нц для i от i1 до i2
| тело цикла (последовательность команд)
кц
```

```
если условие
| то серия 1
| иначе серия 2
все
```

```
если условие
| то серия 1
все
```

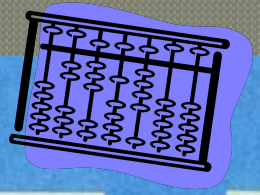
```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
| . . .
| при условие n : серия n
| иначе серия n+1
все
```

```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
| . . .
| при условие n : серия n
все
```

```
утв условие
ввод имена величин
вывод тексты, имена величин, выражения, нс
```

вызов: имя алгоритма( аргументы и имена результатов )

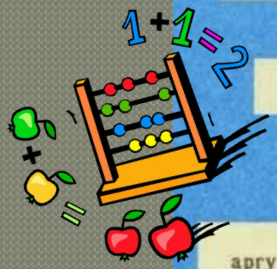
присваивание: имя величины := выражение



## ТИПЫ ВЕЛИЧИН

| | |
|---|---|
| целые | цел |
| вещественные | вещ |
| логические | лог |
| символьные | сим |
| литерные | лит |

| | |
|---|---|
| Таблицы: | |
| целые | цел таб |
| вещественные | вещ таб |
| логические | лог таб |
| символьные | сим таб |



Пример описания: цел i, j, лит t, вещ таб a [1:50]

## ВИДЫ ВЕЛИЧИН

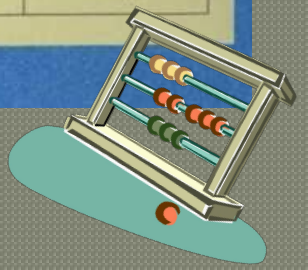
аргументы (арг) - описываются в заголовке алгоритма
результаты (рез) - описываются в заголовке алгоритма
значения функций (знач) - описываются указанием типа перед именем алгоритма-функции
промежуточные - описываются в строке нач алгоритма
общие - описываются после строки исп исполнителя

## ОБЩИЙ ВИД ИСПОЛНИТЕЛЯ

```
исп имя
| описание общих величин исполнителя
| команды для задания начальных значений общих величин
| алгоритмы исполнителя
кон
```

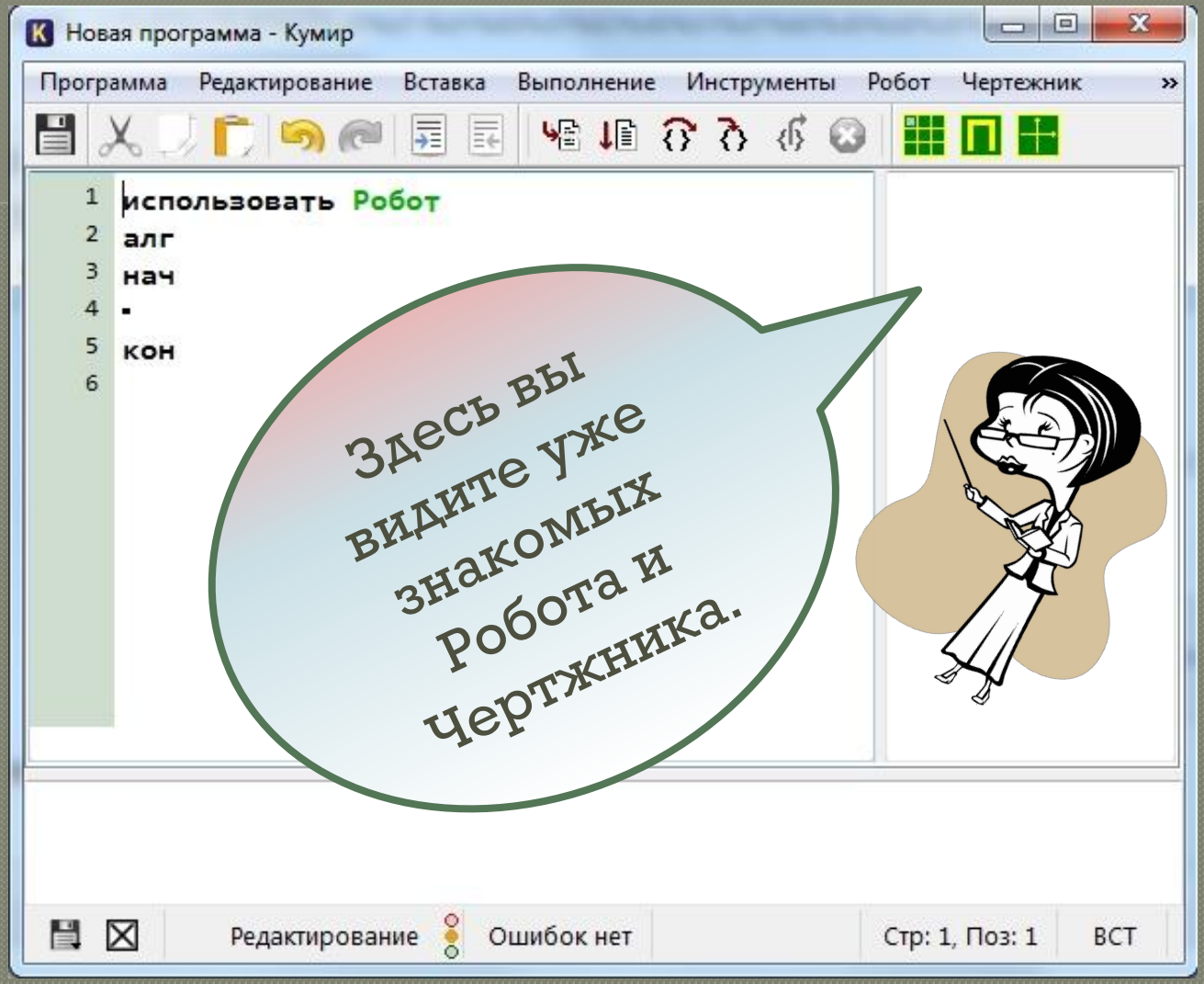
## ОБОЗНАЧЕНИЯ ЗНАКОВ ОПЕРАЦИЙ И СТАНДАРТНЫХ ФУНКЦИЙ

| Название операции или функции | | Форма записи |
|---|---|---|
| сложение | | x + y |
| вычитание | | x − y |
| умножение | | x * y |
| деление | | x / y |
| возведение в степень | | x * * y |
| корень квадратный | | sqrt(x) |
| абсолютная величина | | abs(x) |
| знак числа (−1, 0 или 1) | | sign(x) |
| синус | sin x | sin(x) |
| косинус | cos x | cos(x) |
| тангенс | tg x | tg (x) |
| котангенс | ctg x | ctg(x) |
| арксинус | arcsin x | arcsin(x) |
| арккосинус | arccos x | arccos(x) |
| арктангенс | arctg x | arctg (x) |
| арккотангенс | arcctg x | arcctg(x) |
| натуральный логарифм | ln x | ln(x) |
| десятичный логарифм | lg x | lg(x) |
| степень числа e (e≈2.718281) | $e^x$ | exp(x) |
| минимум из чисел x и y | | min(x,y) |
| максимум из чисел x и y | | max(x,y) |
| остаток от деления x на y | (x,y − целые) | mod(x,y) |
| частное от деления x на y | (x,y − целые) | div(x,y) |
| целая часть числа x | | int(x) |
| случайное число в диапазоне от 0 до x | | rnd(x) |



**Кумир позволяет не только чертить, но и производить математические расчеты.**

# Так выглядит среда Кумира:



**Надеюсь, основная цель достигнута: ГРИС помогли вам сделать плавный переход к изучению программирования.**

- Желаю вам успехо