Графический учебный исполнитель (ГРИС) Стрелочка

Учебными исполнителями называют различные образы на экране компьютера, которыми можно управлять, отдавая команды. Используются они для обучения составлению управляющих алгоритмов.

Одни исполнители создают рисунки на экране, другие складывают слова из кубиков с буквами, третьи перетаскивают предметы из одного места в другое. Все эти исполнители управляются программным путем. Любому из них свойственна определенная ***среда деятельности, система команд управления, режимы работы.*** С помощью каждого из таких исполнителей можно учиться строить алгоритмы управления.

Многие из учебных исполнителей занимаются рисованием на экране компьютера. Пусть наш гипотетический (придуманный) исполнитель тоже будет из этой компании. Назовем его ГРИС, что значит Графический Исполнитель.

Что умеет делать ГРИС? Он может перемещаться по полю и своим хвостом рисовать на этом поле. *Обстановка, в которой действует исполнитель, называется* ***средой исполнителя***.

Среда исполнителя - это лист (страница экрана) для рисования. ГРИС может перемещаться в горизонтальном и вертикальном направлениях с постоянным шагом. На рисунке пунктиром показана сетка с периодом равным шагу исполнителя. Исполнитель может двигаться только по линиям этой сетки. ГРИС не может выходить за границы поля.

Состояние исполнителя на поле определяется, во-первых его местоположением (в какой точке поля он находится) и направлением (куда он смотрит). ГРИС может шагать или прыгать по линиям сетки, а также поворачиваться. Поворачиваться он умеет только против часовой стрелки (налево).

Графический исполнитель — это объект управления. А управлять им будем мы с вами.

Целью управления является получение определенного рисунка. Понятно, что этот рисунок может состоять только из горизонтальных и вертикальных отрезков, в других направлениях ГРИС двигаться не умеет.

Задача обычно ставится так: исполнитель находится в данной точке поля, смотрит в данном направлении. Требуется получить определенный рисунок. Например: ГРИС находится в середине поля и смотрит направо. Нарисовать букву «Т» с длиной линий, равной четырем шагам.

Первоначально исполнителю придается исходное состояние. Это делается в специальном режиме установки (желтое меню, клавиша F3).

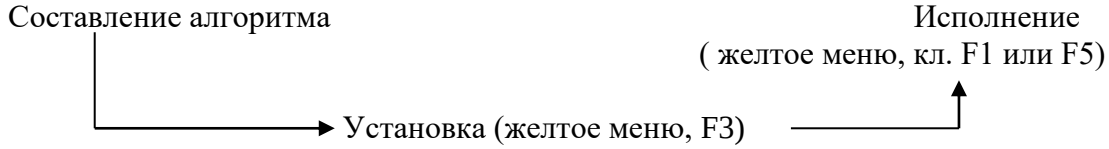
Теперь перейдем к управлению графическим исполнителем. Здесь возможны два режима: ***режим прямого управления*** **и** ***режим программного управления.***

Работа в режиме прямого управления происходит так: человек отдает команду, ГРИС ее выполняет; затем отдается следующая команда и т.д

В режиме прямого управления система команд исполнителя следующая:
шаг - перемещение ГРИС на один шаг вперед с рисованием линии;
поворот - поворот на 90^0 против часовой стрелки (налево);
прыжок - перемещение на один шаг вперед без рисования линии.
Эти команды будем называть *простыми командами.*

Работа в программном режиме имитирует автоматическое управление исполнителем. Управляющая система (компьютер) обладает памятью, в которую заносится программа. *Человек составляет программу и вводит ее в память.* Затем ГРИС переводится в режим установки и человек вручную (с помощью определенных клавиш) устанавливает исходное состояние исполнителя. После этого производится переход в режим выполнения и ГРИС начинает работать по программе. Если возникает ситуация, при которой он не может выполнить очередную команду (выход за границу поля), то выполнение программы завершается аварийно. Если аварии не происходит, то работа исполнителя заканчивается на последней команде.

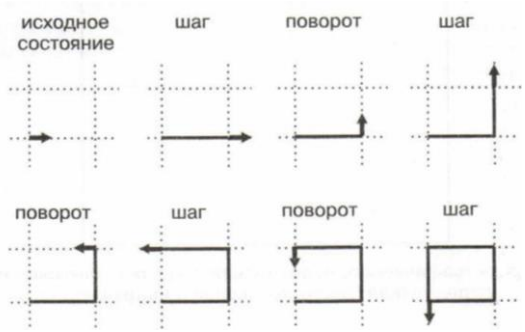
Таким образом, программное управление графическим исполнителем проходит этап подготовки (запись алгоритма и установка исходного состояния) и этап исполнения программы (желтое меню, кл F1или F5).



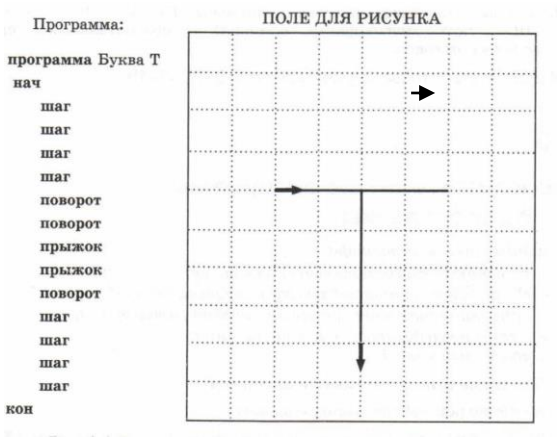
В режиме программного управления по-прежнему используются команды шаг, поворот, прыжок. Однако в этом режиме есть еще и другие команды. С ними вы познакомитесь позже.

Язык программирования для графического исполнителя - это учебный алгоритмический язык (АЯ). Поэтому алгоритмы управления ГРИСом, записанные на АЯ, являются для него одновременно и программами.

Например, пусть требуется нарисовать квадрат со стороной, равной одному шагу. Исходное положение ГРИС – в левом нижнем углу квадрата, направление - на восток. Будем отмечать состояние исполнителя маленькой стрелкой. Тогда последовательность команд и результаты их выполнения будут следующими:



Будем осваивать программирование на примерах решения конкретных задач. С новыми командами СКИ будем знакомиться по мере появления потребности в них.



Задача 1. Составим и выполним программу, по которой ГРИС нарисует на поле букву «Т». Пусть длина вертикального и горизонтального отрезков равна четырем шагам.

Исходное состояние — чистый лист. Исполнитель – в точке, где будет находится левый конец горизонтального отрезка, направление — направо.

Стрелка указывает конечное состояние исполнителя

Структура такого алгоритма называется - *линейной*. Команды выполняются по очереди, каждая только один раз.

Линейный алгоритм – алгоритм в котором команды выполняются строго друг за другом в порядке их следования (записи).

Для решения этой задачи оказалось достаточно той части СКИ, которая используется в режиме прямого управления.

Коротко о главном

ГРИС — это графический исполнитель, назначение которого - получение чертежей, рисунков на экране дисплея.

Управление ГРИС может происходить в режиме ***прямого управления*** или в режиме ***программного управления.***

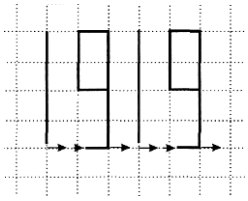
С помощью команд ***шаг, поворот, прыжок*** в пределах рабочего поля можно построить любой рисунок, состоящий из вертикальных и горизонтальных отрезков. Структура управляющего алгоритма при этом будет ***линейной.***

Вопросы и задания

1. Какую работу может выполнять ГРИС?
2. Что представляет собой среда исполнителя ГРИС?
3. В чем разница между управлением в прямом режиме и в программном режиме?
4. Какие простые команды входят в СКИ ГРИС; как они выполняются?
5. В какой последовательности происходит выполнение команд в линейном алгоритме?
6. Может ли исполнитель нарисовать: прямоугольник, треугольник, пятиконечную звезду, буквы «Н», «Х», «Р», «М»?
7. Составить программы рисования символов: «Е», «П», «Б», «Ч»,«Ц», «Ш», а также других фигур, состоящих из горизонтальных и вертикальных отрезков.

§2. Вспомогательные алгоритмы и подпрограммы

А сейчас решим следующую задачу.



Задача 2. Пусть требуется составить программу, по которой ГРИС напишет на экране четырехзначное число 1919. Поскольку здесь дважды повторяются цифры 1 и 9, нельзя ли сократить работу, написав программу рисования той и другой цифры только один раз? Это действительно можно сделать.

Алгоритм, по которому решается некоторая подзадача из основной задачи и который, как правило, выполняется многократно, называется вспомогательным алгоритмом.

```
программа Число 1919
нач
   сделай ЕДИНИЦА
   прыжок
   сделай ДЕВЯТЬ
   прыжок
   сделай ЕДИНИЦА
   прыжок
   сделай ДЕВЯТЬ
кон
```

В языках программирования их называют подпрограммами или *процедурами.* Последнее название будем использовать в языке управления графическим исполнителем.

Вспомогательный алгоритм (подпрограмма, процедура) – самостоятельный алгоритм, имеющий имя и использующийся в других алгоритмах.

В таком случае алгоритм решения поставленной задачи разделяется на основную программу (основной алгоритм) и процедуры (вспомогательные алгоритмы). Каждая процедура должна иметь свое уникальное имя. Для рассматриваемой задачи имена процедур выберем следующими: ЕДИНИЦА и ДЕВЯТЬ.

Тогда в основном алгоритме **команда обращения** к этим процедурам будут такими:

сделай ЕДИНИЦА
сделай ДЕВЯТЬ

По этим командам управление передается соответствующим процедурам и после их выполнения управление вернется к следующей команде основной программы.

Договоримся, что начальное и конечное состояния ГРИС при вычерчивании каждой цифры будет таким, как показано стрелками на рисунке (внизу, направо). У единицы начальное и конечное состояния совпадают.

Основная программа:

```
алг число 1919
      сделай ЕДИНИЦА
      прыжок
      сделай ДЕВЯТЬ
```

прыжок
сделай ЕДИНИЦА
прыжок
сделай ДЕВЯТЬ

Теперь надо «объяснить» исполнителю, что такое ЕДИНИЦА и что такое ДЕВЯТЬ. Это делается в *описаниях процедур* (см. ниже; порядок выполнения — по столбцам).

процедура ЕДИНИЦА
поворот
шаг
шаг
шаг
шаг
поворот
поворот
прыжок
прыжок
прыжок
прыжок
поворот
конец процедуры

процедура ДЕВЯТЬ
шаг
поворот
шаг
шаг
шаг
шаг
поворот
шаг
поворот
шаг
шаг
поворот
шаг
поворот
поворот
поворот
прыжок
прыжок
поворот
конец процедуры

Данный пример познакомил вас с новой командой из СКИ графического исполнителя — командой *обращения к процедуре***.** Ее формат, то есть общий вид, следующий:

сделай < имя процедуры>

Определение процедуры в алгоритме называется ее *описанием*. Формат описания процедуры:
процедура <имя процедуры>
<тело процедуры>
конец процедуры

Имя в описании и имя в обращении должны точно совпадать (никаких склонений по падежам!). Описание процедур располагается после основного алгоритма.

Добавив к алгоритму описание процедуры, мы тем самым расширили систему команд исполнителя. В данном алгоритме стало возможным использование команды обращения к этой процедуре.

Использованный нами подход облегчает программирование сложных задач. Задача разбивается на более простые подзадачи. Решение каждой оформляется в виде вспомогательного алгоритма, а основной алгоритм организует связку между ними.

Метод программирования, при котором сначала пишется основная программа, затем в ней записываются обращения к пока еще не составленным подпрограммам, а потом описываются эти подпрограммы, называется ***методом последовательной детализации.*** Причем количество шагов детализации может быть гораздо большим, чем в нашем примере, поскольку сами подпрограммы могут содержать внутри себя обращения к другим подпрограммам.

Возможен и другой подход к построению сложных программ: первоначально составляется множество подпрограмм, которые могут понадобиться при решении задачи, а затем пишется основная программа, содержащая обращения к ним. Часто в литературе по программированию используется такая терминология: метод последовательной детализации называют ***программированием сверху вниз,*** а сборочный метод — ***программированием снизу вверх.***

Коротко о главном

Для упрощения программирования сложных задач используются вспомогательные алгоритмы.

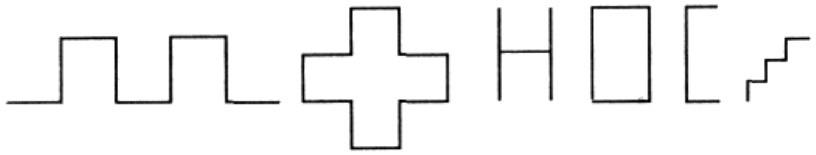
Вспомогательный алгоритм (подпрограмма, процедура) – самостоятельный алгоритм, имеющий имя и использующийся в других алгоритмах.

Вспомогательный алгоритм должен быть описан. После этого в основном алгоритме можно использовать команду обращения к этому вспомогательному алгоритму.

Метод программирования, при котором сначала записывается основной алгоритм, а затем описываются использованные в нем вспомогательные алгоритмы, называется методом последовательной детализации, или программированием сверху вниз. Обратный порядок программирования называется программированием снизу вверх.

Вопросы и задания

1. Что такое основной алгоритм; вспомогательный алгоритм?
2. Чем отличается описание вспомогательного алгоритма от обращения к вспомогательному алгоритму?
3. Каковы правила описания вспомогательных алгоритмов (процедур) для исполнителя ГРИС?
4. Как записывается команда обращения к процедуре в языке исполнителя ГРИС?
5. В чем суть метода последовательной детализации?
6. Что такое программирование снизу вверх, сверху вниз?
7. Используя вспомогательные алгоритмы, нарисовать:



§3. Циклические алгоритмы

Обсудим решение следующей задачи.

Задача 3. Исходное положение: ГРИС у левого края поля направление — направо. Требуется нарисовать горизонтальную линию через весь экран.

Задачу можно решить, написав 15 раз команду шаг (если поперек поля рисунка 15 шагов). Но есть и более короткий вариант программы:

```
пока впереди не край, повторять
    шаг
конец цикла
```

Здесь использована команда, которая называется ***циклом***.

Циклический алгоритм *– это алгоритм в котором команда или группа команд выполняются многократно, пока истинно условие.*

Формат команды цикла следующий:

```
пока <условие>, повторять
    <тело цикла>
кц
```

Служебное слово **кц** обозначает конец цикла.

Условие *– это выражение принимающее значение истина («да») или ложь («нет»)*

Это первая команда из СКИ, которая использует обратную связь между графическим исполнителем и управляющим им компьютером. Она заключается в том, что проверяется, не вышел ли ГРИС на край поля и не грозит ли ему следующий шаг или прыжок в этом направлении аварией? Проверяемые **условия** звучат так: ***«впереди край?»*** или ***«впереди не край?».*** На что машина получает от исполнителя ответ *«да»* или *«нет».*

В приведенном примере проверяется условие «Впереди не край?». Если «да», то делается шаг (то есть выполняется <тело цикла>).

Тело цикла – команда или группа команд повторяющихся многократно.

Затем происходит возврат на проверку условия, и все повторяется. Если проверка условия дает отрицательный результат (то есть впереди — край), то выполнение цикла завершится и будет исполняться следующая команда программы.

При программировании цикла важно думать о том, чтобы цикл был конечным. Цикл, записанный выше, — конечный. Двигаясь в одном направлении, исполнитель обязательно достигнет края и на этом выполнение цикла закончится.

Ситуация, при которой выполнение цикла никогда не заканчивается, называется зацикливанием. Пусть ГРИС находится в середине поля. Исполнение следующего цикла:

шаг
поворот
КЦ

никогда не закончится. ГРИС будет бесконечно рисовать квадратик, т.к. проверка условия «впереди не край?» всегда будет давать положительный результат.

<u>Задача 4.</u> Теперь составим алгоритм, по которому графический исполнитель нарисует прямоугольную рамку по краю поля. Исходное положение: ГРИС находится в левом верхнем углу, смотрит вниз.

Рамка состоит из четырех линий, поэтому разумно воспользоваться процедурой, проводящей линию от края до края поля. Опять будем действовать методом последовательной детализации. Напишем сначала основную программу.

алг Рамка

сделай ЛИНИЯ
поворот
сделай ЛИНИЯ
поворот
сделай ЛИНИЯ
поворот
сделай ЛИНИЯ

Алгоритм проведения линии нами уже рассматривалась. Осталось оформить ее в виде процедуры.

процедура ЛИНИЯ
пока впереди не край **, повторять**
шаг
конец цикла
конец процедуры

При составлении этого алгоритма использовалась одношаговая детализация в такой последовательности: **ОСНОВНОЙ АЛГОРИТМ** ⟶ **процедура ЛИНИЯ**
шаг детализация

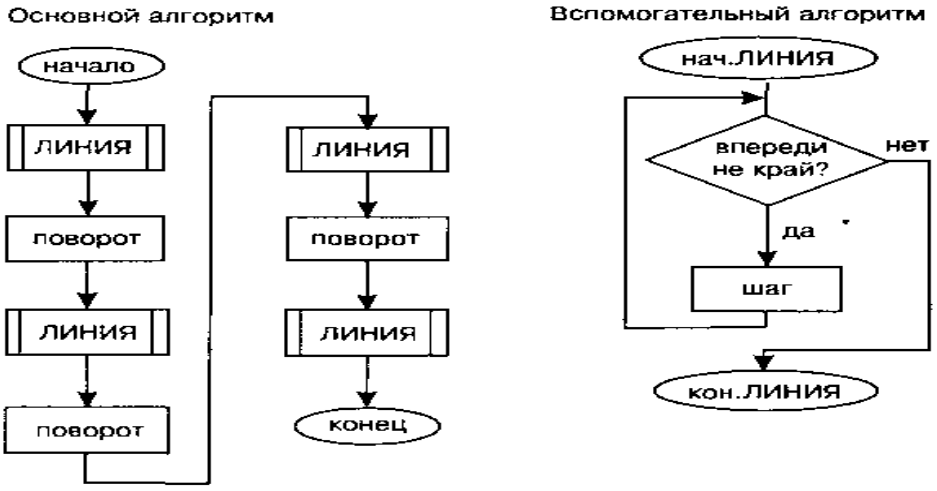
Блок-схемы алгоритмов

Начиная с 50-х годов, т.е. еще с эпохи ЭВМ первого поколения, программисты стали использовать графические схемы, изображающие алгоритмы, которые получили название блок-схем.

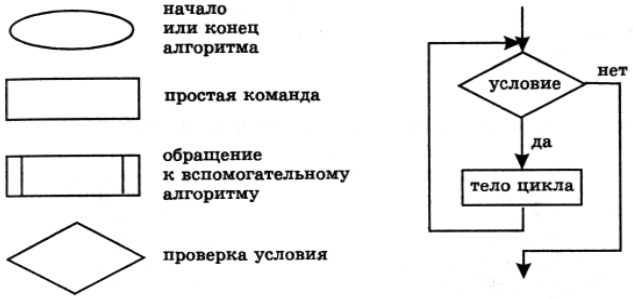
Блок-схема состоит из фигур (блоков), обозначающих отдельные действия исполнителя, и стрелок, соединяющих эти блоки и указывающих на последовательность их выполнения. Внутри каждого блока записывается выполняемое действие. Сама форма блока подсказывает характер операции, которую он обозначает. Для придания наглядности и единообразия схемам алгоритмов все графические элементы стандартизированы.

Посмотрите на рисунок , где показана блок-схема алгоритма рисования рамки. Она состоит из двух частей: блок-схемы основного алгоритма и блок-схемы вспомогательного алгоритма ЛИНИЯ.

Из этих схем видно назначение блоков различной формы.



Команда цикла изображается не отдельным блоком, а целой структурой. Такую структуру называют *циклом с предусловием* (так как условие предшествует телу цикла). Есть и другой вариант названия: ***цикл-пока*** (пока условие истинно, повторяется выполнение тела цикла).



При решении следующей задачи снова будем использовать метод последовательной детализации.

Задача 5. Требуется расчертить экран горизонтальными линиями. Исходное состояние исполнителя: верхний левый угол, направление — вниз.

В алгоритме для решения этой задачи используется та же процедура ЛИНИЯ. Другая процедура — ВОЗВРАТ — возвращает ГРИС к исходному положению для рисования следующей линии.

Алг Разлиновка

пока впереди не край, **повторять**
поворот
сделай ЛИНИЯ
сделай ВОЗВРАТ
прыжок
конец цикла
поворот
сделай ЛИНИЯ

процедура ВОЗВРАТ

поворот
поворот
пока впереди не край, **повторять**
прыжок
конец цикла
поворот
конец процедуры

Коротко о главном

Для программирования повторяющихся действий применяется команда цикла, которая имеет следующую структуру:

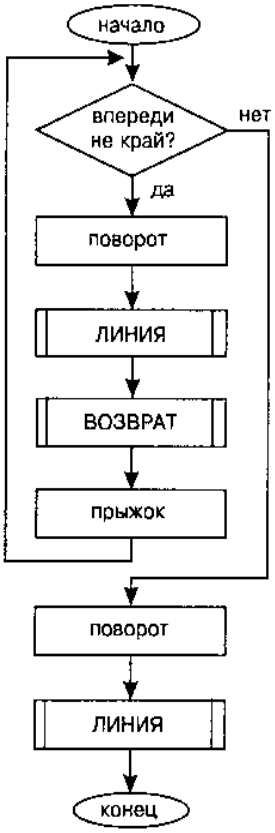
пока <условие>, **повторять**
<тело цикла>
конец цикла

Если проверяемое условие выполняется (истинно), то выполняются команды, составляющие тело цикла. Если условие ложно, то происходит выход из цикла.

При программировании цикла необходимо следить за тем, чтобы не допускалось зацикливания.

Блок-схема — это графический способ описания алгоритма. Блоки обозначают действия исполнителя, а соединяющие их стрелки указывают на последовательность выполнения действий.

Блок-схемы основного и вспомогательного алгоритмо



Вопросы и задания

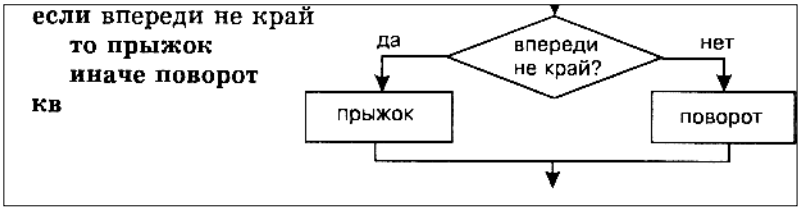
1. Что такое цикл? Как записывается команда цикла?
2. Что такое условие цикла? Что такое тело цикла?
3. В каком случае происходит зацикливание алгоритма?
4. Что такое блок-схема?
5. Из каких блоков составляются блок-схемы (как они изображаются и что обозначают)?
6. Что обозначают стрелки на блок-схемах?
7. Составить алгоритм, переводящий ГРИС в угол поля из любого исходного состояния.
8. Составить алгоритм рисования прямоугольной рамки вдоль края поля, исходя из любого начального состояния исполнителя.

§4. Ветвление и последовательная детализация алгоритма

Команда ветвления. Познакомимся еще с одной командой ГРИС. Она называется *командой ветвления.* Формат команды ветвления такой:

если <условие> **то**
<серия **1**>
иначе
<серия **2**>
конец ветвления

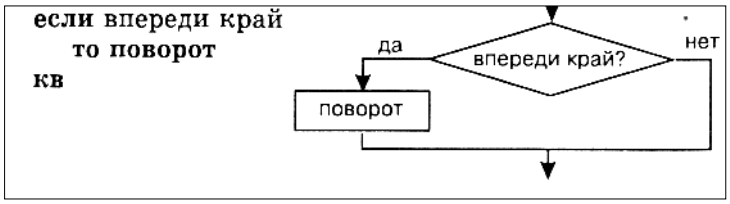
По-прежнему ГРИС может проверять только два условия: *«впереди край?» или «впереди не край?».* <Серия> - это одна или несколько следующих друг за другом команд. Если <условие> справедливо, то выполняется <серия 1>, в противном случае — <серия 2>. Такое ветвление называется ***полным.***



В некоторых случаях используется ***неполная форма*** команды ветвления. Неполная команда ветвления имеет следующий формат:

если <условие> **то**
<серия>
конец ветвления

Здесь <серия> выполняется, если <условие> справедливо.

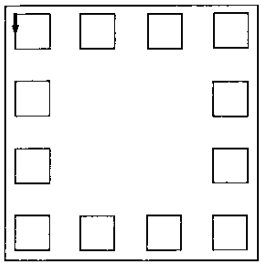


Составим последнюю, сравнительно сложную программу для ГРИС. На этом примере вы увидите, что применение метода последовательной детализации облегчает решение некоторых головоломных задач.

Задача 6. Построить орнамент, состоящий из квадратов по краю поля. Исходное положение ГРИС — в верхнем левом углу, направлен вниз.

Цепочку квадратов, нарисованных от края до края поля, назовем РЯД. Ниже будет составлена процедура, рисующая ряд квадратов. Но сначала напишем основную программу (исходное положение показано стрелкой).

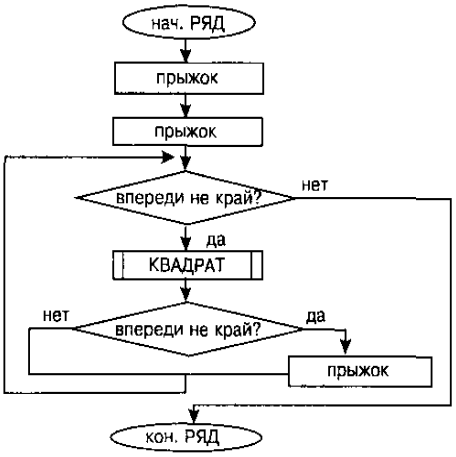
Алг Орнамент
Сделай Ряд
Поворот
Сделай Ряд
Поворот
Сделай Ряд
Поворот
Сделай ряд



Ниже приводится процедура Ряд. В ней, в свою очередь, имеется обращение к другой процедуре, которая называется КВАДРАТ. Из названия ясно, что по этой процедуре чертится один квадрат.

процедура РЯД
прыжок
прыжок
пока впереди не край**, повторять**
сделай КВАДРАТ
если впереди не край **то**
прыжок
конец ветвления
конец цикла
прыжок
конец процедуры

процедура КВАДРАТ
шаг
поворот
шаг
поворот
шаг
поворот
шаг
поворот
конец процедуры



В процедуре РЯД в теле цикла содержится неполное ветвление. Структуру такого алгоритма можно назвать так: *цикл с вложенным ветвлением.*

Составление этой программы потребовало два шага детализации алгоритма, которые выполнялись в такой последовательности:

ОСНОВНОЙ АЛГОРИТМ ⟶ **процедура РЯД** ⟶ **процедура** КВАДРАТ
1-й шаг детализации 2-й шаг детализации

Теперь вам известны все команды управления графическим исполнителем. Их можно разделить на три группы: простые команды; команда обращения к процедуре; структурные команды. К третьей группе относятся команды цикла и ветвления.

Коротко о главном

Структурная команда ветвления имеет следующий формат:

если <условие> **то** <серия 1>
иначе <серия 2>
конец ветвления

Если <условие> истинно, то выполняются команды, составляющие серию 1, если ложно, то - серия 2. <Серия 1> называется положительной ветвью ветвления, <серия 2> - отрицательной ветвью.

Неполная команда ветвления имеет следующий формат:

если <условие> **то**
<серия >
конец ветвления

Если условие истинно, то выполняется <серия>, если ложно, то сразу происходит переход к следующей команде алгоритма.

Сложные алгоритмы удобно строить путем многошаговой Детализации.

Вопросы и задания

1. Что такое многошаговая детализация?
2. Из каких команд могут состоять вспомогательные алгоритмы последнего уровня детализации?
3. Какой формат имеет команда ветвления? Какие действия исполнителя она определяет?
4. Чем отличается полное ветвление от неполного?
5. Путем многошаговой детализации составить алгоритм для решения следующих задач:
   - Расчертить все поле горизонтальными пунктирными линиями;
   - Нарисовать квадраты во всех четырех углах поля;
   - Расчертить все поле в клетку со стороной, равной шагу.