А.Г.Гейн

**Методика изучения алгоритмизации с помощью учебных исполнителей**

Книга для учителя

Екатеринбург

2011

**Содержание**

Введение ………………………………………………………………………. 3

Дидактические цели и методические установки введения понятия «формальный исполнитель» ….…………………………………………. 7

Методика изучения алгоритмических конструкций с помощью исполнителя Паркетчик ………………………………………………………………… 20

Методика изучения переменных и структур данных с помощью исполнителя Паркетчик ………………………………………………………………… 31

Литература ……………………………………………………………………. 43

**Введение**

Появление более чем четверть века назад в школьном расписании предмета «Информатика» (а точнее «Основы информатики и вычислительной техники») одним из главных мотивов имело целенаправленное развитие у школьников алгоритмического мышления. Именно это позволило в то время несколько лет вести преподавание данного курса в так называемом бескомпьютерном варианте, что сегодня представляется абсолютно абсурдным. Коренные перемены в обществе и стремительное вхождение в нашу жизнь информационных технологий привело к тому, что в 90-е годы школьный курс информатики трансформировался в изучение офисных и близких к ним технологий, поскольку именно владение такими технологиями давало выпускнику школы шанс найти приемлемое место в бурно информатизирующемся обществе. Взаимодействие с компьютером заняло центральное место в идеологии курса информатики того времени (1995 – 2005 гг.). Такое взаимодействие, как и взаимодействие с любым инструментом – будь то токарный станок или швейная машинка, технологично, и потому обучение в курсе информатики неизбежно уходило в изучение соответствующих технологий. Отличие компьютера от таких инструментов, как токарный станок или швейная машинка, в этом плане (разумеется, у него есть много других отличий!) состоит в том, что программное обеспечение, посредством которого и реализуется взаимодействие человека с компьютером, делится на прикладное и системное.[[1]](#footnote-1) Потому и освоение технологий здесь также раздваивается – обучение технологиям работы с прикладными программами и обучение технологии программирования. Более того, обучение программированию стали выдавать за обучение алгоритмизации. Конечно, в Федеральном компоненте государственного общеобразовательного стандарта, принятого в 2004 г., наряду с технологическим контентом присутствуют все основные дидактические линии курса информатики, но то мизерное количество часов, отводимое на изучение курса «Информатика и ИКТ», и, как уже было сказано, наследие предшествующего десятилетия, де факто сводили на нет освоение собственно информатики как дисциплины, рассматривающей фундаментальные аспекты протекания информационных процессов. Их изучение в лучшем случае осуществлялось скороговоркой на фоне массированного изучения текстового редактора Word, табличного процессора Excel, какого-либо графического редактора типа Corel Draw или Adobe Photoshop, средств презентационной графики, скажем, Power Point, и т.д. Проводимое нами с 2006 г. анкетирование студентов, поступивших на 1 курс математико-механического факультета Уральского государственного университета им. А.М. Горького (ныне вошедшего в состав Федерального университета им. первого Президента России Б.Н.Ельцина), показывает, что более 90% говорят о своем школьном курсе информатике именно как предмете, на котором изучались указанные выше средства информационных технологий (иногда, если речь идет о специализированных классах, еще добавляются языки программирования). Традиционная для нашей страны ситуация, когда строгость закона (а государственный образовательный стандарт имеет статус закона) смягчается необязательностью его исполнения.

Подмена освоения алгоритмизации изучением программирования имеет еще один отрицательный момент. Программирование – это вид профессиональной деятельности, которой по данным Федеральной службы государственной статистики (Росстат) на 2009 год занимается менее 2% всех работающих. Даже для высоко развитых в информационном отношении стран доля занятых в сфере ИТ-индустрии составляет 3,74% в США, 3,16% в Великобритании и 3,14% в Германии (см. [5]). Вряд ли для нашей страны следует ожидать в ближайшем будущем скачка даже до 3%. Поэтому для подавляющего большинства школьников обучение программированию совершенно справедливо представляется абсолютно ненужным и бесперспективным занятием, навязанным свыше бездушными чиновниками из Министерства образования и науки.

Федеральный государственный образовательный стандарт (ФГОС) второго поколения включает в себя лишь рамочное решение вопроса о содержании курса информатики. Вот как оно представлено в описании предметных результатов освоения основной образовательной программы основного общего образования для единой предметной области «Математика и информатика» ([6], с. 14 – 15):

* развитие умений применять изученные понятия, результаты, методы для решения задач практического характера и задач из смежных дисциплин с использованием при необходимости справочных материалов, компьютера, пользоваться оценкой и прикидкой при практических расчётах;
* формирование информационной и алгоритмической культуры; формирование представления о компьютере как универсальном устройстве обработки информации; развитие основных навыков и умений использования компьютерных устройств;
* формирование представления об основных изучаемых понятиях: информация, алгоритм, модель – и их свойствах;
* развитие алгоритмического мышления, необходимого для профессиональной деятельности в современном обществе; развитие умений составить и записать алгоритм для конкретного исполнителя; формирование знаний об алгоритмических конструкциях, логических значениях и операциях; знакомство с одним из языков программирования и основными алгоритмическими структурами — линейной, условной и циклической;
* формирование умений формализации и структурирования информации, умения выбирать способ представления данных в соответствии с поставленной задачей — таблицы, схемы, графики, диаграммы, с использованием соответствующих программных средств обработки данных;
* формирование навыков и умений безопасного и целесообразного поведения при работе с компьютерными программами и в Интернете, умения соблюдать нормы информационной этики и права.

Развитие алгоритмического мышления, в том числе с умениями составлять алгоритмы для конкретных исполнителей, прямо стоит в целевых предметных установках нового стандарта. Впрочем, об освоении алгоритмов, формировании алгоритмической культуры речь идет и в других его пунктах.

Современная концепция российского образования делает акцент не только на освоение знаний, умений и навыков, но и формирование компетенций, наличие которых свидетельствует о том, что учащиеся готовы к применению полученных ими знаний и умений для решения жизненных задач, выходящих за рамки репродуктивного уровня. В частности, такими компетенциями должны быть сформированы у ученика по отношению к продуктам информационной деятельности (как созданным им самим, так и другими людьми), к способам обмена этими продуктами, к способам их хранения, а также по отношению к техническим и программным средствам информационной деятельности. На наш взгляд, эти компетенции развиваются у учащихся постепенно, проходя следующие уровни:

* *уровень исполнительского владения компетенциями*: умение точно и правильно создавать информационный продукт или совершать над ним заданную операцию по известной схеме, образцу;
* *уровень технологического владения компетенциями*: умение самому спланировать, придумать схему создания информационного продукта или операций над ним;
* *уровень экспертного владения компетенциями*: умение дать обоснованную качественную оценку информационному продукту, указав его достоинства и недостатки;
* *уровень аналитико-синтезирующего владения компетенциями*: умение на основе анализа готового информационного продукта и технологии обращения с ним предлагать изменения в структуре самого продукта или технологии его изготовления.

Разработка алгоритмов и написание программ предоставляет прекрасный полигон для формирования указанных компетенций на всех уровнях. Мы в дальнейшем будем специально останавливаться на возможностях формирования компетенций при обучении алгоритмизации и программированию.

**Дидактические цели и методические установки введения понятия «формальный исполнитель»**

Дидактическая линия, связанная с изучением алгоритмов, является, как уже было сказано, одной из главных в школьном курсе информатики. Нам представляется, что изучение материала этой линии необходимо начинать с обсуждения базового понятия информатики понятия «формальный исполнитель». Выдвижение в этой теме на первый план понятия исполнителя, а не понятия алгоритма продиктовано комплексом дидактических причин. Мы укажем три из них, наиболее, на наш взгляд, важные.

Во-первых, обучение алгоритмизации должно рассматриваться не как промежуточный шаг в обучении программированию, а как средство развития мышления. Поэтому обучение алгоритмизация должна выполняться не на искусственном языке программирования, ориентированном на общение с компьютером, а на естественном языке, на котором человек может описывать и свою собственную алгоритмизуемую деятельность.[[2]](#footnote-2)

Во-вторых, компьютер вовсе не является единственным программируемым устройством. Поэтому изучать алгоритмизацию с постоянной ориентацией на компьютер (а точнее, опять-таки на конкретный язык программирования) значит резко сузить реальную практику алгоритмических построений. Но как только мы начинаем представлять себе, что программируется не только компьютер, сразу возникает вопрос о списке тех управляющих воздействий, передаваемых с помощью команд, которые допустимы для данного программируемого устройства.

В-третьих, общность алгоритмических конструкции, не зависящая от того, для какого исполнителя создается алгоритм, при наличии (хотя бы потенциально) достаточного разнообразия формальных исполнителей, показывает учащимся, что изучение алгоритмизации является общеобразовательной, а не узкоспециальной (программистской) ценностью.

Знакомство с понятием исполнителя естественно начинать с введения следующих понятий: формальный исполнитель, его система команд, система его допустимых действий и достижимые цели. Этого достаточно, чтобы затем уже быстро и доступно ввести понятия алгоритма и программы. И здесь важно, чтобы учащиеся, разумеется, при участии учителя, привели достаточно широкий список примеров разнообразных формальных исполнителей (или объектов, которые в определенных условиях ведут себя как формальные исполнители).

Каждый исполнитель полностью определен системой своих допустимых действий и средой, в которой он функционирует. Поэтому обсуждая примеры формальных исполнителей сразу же надо требовать от учащихся точного определения ими взаимодействия исполнителя с окружающей средой и списка

Учащиеся склонны думать, что формальным исполнителем всегда выступает автоматическое устройство, например, компьютер. На самом деле и человек довольно часто выступает в роли формального исполнителя. Важно, чтобы учащиеся осознали, что управляет ими не только *внешний* субъект (учитель, родитель, староста класса, друг и т.д.) или объект (например, светофор), хотя и здесь должно быть достигнуто достаточное разнообразие примеров, но и каждый управляет собой сам, планируя свои действия ежедневно, ежечасно, а может быть и ежеминутно.

Методологическое значение изучения понятия «формальный исполнитель» трудно переоценить. На самом деле именно это понятие сделало возможным получать ответ на такой вопрос, существует ли алгоритм, позволяющий решить ту или иную задачу. Мы же остановимся лишь на использовании этого понятия как дидактического средства при изучении алгоритмизации. При этом, как уже было отмечено выше, основное внимание удается сосредоточить именно на развитии у учащихся алгоритмического стиля мышления и проверки понимания именно алгоритмической сущности тех или иных процессов, а не программистской оболочки. Не случайно, и в ГИА, и в ЕГЭ значительное число заданий по алгоритмизации сформулировано именно с использованием исполнителей, а не в форме программ на языке программирования (хотя задания такого типа, разумеется, тоже присутствуют). Впрочем, умение эффективно выполнять такие задания – это лишь надводная часть айсберга. Исполнители в учебном процессе появились, прежде всего, для обучения школьников алгоритмизации.

Использование учебных исполнителей при изучении алгоритмизации имеет целый ряд преимуществ перед изучением напрямую языка программирования, и мы являемся сторонниками активного использования учебных исполнителей алгоритмов при обучении информатике. Постараемся обосновать свою точку зрения.

1. Основная цель курса информатики, каким мы его видим научить школьников решать задачи с помощью компьютера. При этом принципиально важно, что для решения каждого класса задач требуется свой исполнитель. “Научить” компьютер решать задачи данного класса это значит сымитировать на нем допустимые действия исполнителя. Можно сказать, что каждое применение компьютера основывается на том, что с его помощью можно имитировать различные наборы допустимых действий. Понимание школьниками этого обстоятельства лучше всего достигается при использовании в курсе информатики учебного исполнителя, имитированного на компьютере.
2. Исполнители, сымитированные на компьютере, фактически играют в курсе информатики роль технических средств обучения, предоставляя учителю богатые возможности для тех или иных методических решений.
3. Учащиеся привыкли к алгоритмам, предназначенным для человека, которые можно записывать в произвольной форме, не фиксируя список допустимых действий. В то же время при работе на компьютере необходима и строгость в записи организации действий (т.е. в записи алгоритмических конструкций), и понимание ограниченности и семантической однозначности набора допустимых действий. Но для компьютера невозможно указать его полный набор допустимых действий! Применение исполнителей позволяет обойти эту трудность естественным путем.
4. Простые и наглядные исполнители допускают имитацию не только на компьютере, но и, как правило, на классной доске. Это важное подспорье учителю при решении задач на составление алгоритмов, предваряющее выход с готовым алгоритмом на машину. Вообще, надо стремиться к тому, чтобы у школьников не возникала порочная практика составления программ прямо за компьютером.
5. Применение исполнителей в процессе изучения алгоритмизации позволяет делать это в “рафинированном” виде, не отвлекая учащихся на языковые особенности того или иного языка программирования.
6. Обычно противники использования исполнителей утверждают, что им хватает лишь одного исполнителя компьютера. Это утверждение не вполне корректно, поскольку неявно подразумевается тот или иной язык программирования или стандартный набор информационных технологий. Фактически это те же исполнители, только с трудно обозримым набором допустимых действий.
7. Наконец, при обучении школьника программированию лишь на каком-то одном конкретном языке невольно создает стереотип языка общения с компьютером, и при переходе к другому языку возникают значительные трудности. Наличие разнообразных исполнителей помогает преодолеть их.

Идея применения в обучении школьников компьютерно имитированных учебных исполнителей принадлежит американскому ученому и педагогу С.Пейперту. Придуманные им Черепашка и язык ЛОГО, предназначенный для управления этим исполнителем, триумфально прошествовали по всему миру и в настоящее время продолжают оставаться одним из наиболее востребованных средств обучения алгоритмизации. В России традиция активного использования учебных исполнителей в обучении школьников программированию была заложена Г.А. Звенигородским. Параллельно та же идея использовалась А.Г. Кушниренко для обучения программированию студентов мехмата МГУ; позже она была перенесена в школьные учебники, разработанные под его руководством, и представлена сейчас в системе КуМир.

Одним из учебных исполнителей является Паркетчик,[[3]](#footnote-3) который уже более 20 лет активно используется в учебниках информатики, созданных авторским коллективом под руководством профессора А.Г. Гейна (см., например, [1 – 3].[[4]](#footnote-4) Этот исполнитель был разработан на основе анализа существовавших до него учебных исполнителей, предложенных С. Пейпертом, Г.А. Звенигородским, А.Г. Кушниренко и др., а его применение для обучения алгоритмизации впервые было описано в учебнике [4]. Как и учебные исполнители упомянутых выше авторов, Паркетчик обладает важными дидактическими достоинствами: наглядность в исполнении алгоритмов, русскоязычный интерфейс, детальная диагностика синтаксических и семантических ошибок. В то же время он подчеркнуто минимален по набору допустимых действий, дабы учащиеся не отвлекались на детали, несущественные с точки зрения обучения алгоритмизации. Важной особенностью исполнителя Паркетчик является возможность наглядного моделирования работы со структурами данных различного типа – массивом, стеком, очередью, графом. Кроме того, Паркетчик позволяет моделировать машину Поста (разумеется, на ленте ограниченной длины), что дает возможность в классах углубленного изучения информатики затрагивать весьма сложные вопросы теории алгоритмов.

Дидактическое предназначение Паркетчика обучение алгоритмическим конструкциям и способам организации данных. Поэтому при разработке этого исполнителя мы стремились свести к минимуму разнообразие допустимых действий (чтобы не перегружать память ученика) и не навязывать ему функции, выходящие за рамки поставленной дидактической цели. В этом отношении Черепашка ЛОГО умеет делать «слишком много», и освоение того, что она умеет, нередко уводит в сторону от главной задачи обучения алгоритмизации.

Средой (для детей можно сказать, игровым полем) Паркетчика являются лист клетчатой бумаги (точнее, его изображение на экране компьютера), а манипулирует он квадратными плитками двух цветов — красного и зеленого. Каждая плитка покрывает в точности одну клетку на бумаге. Паркетчик занимается тем, что, исполняя те или иные алгоритмы, выкладывает на листе бумаги орнаменты (паркеты) из этих плиток, для простоты можно считать, что у Паркетчика неограниченный запас красных и зеленых плиток (хотя ясно, что ему не надо плиток каждого цвета больше, чем клеток на поле, где он работает). На рисунке 1 показано поле Паркетчика с выложенным столбцом из красных плиток.

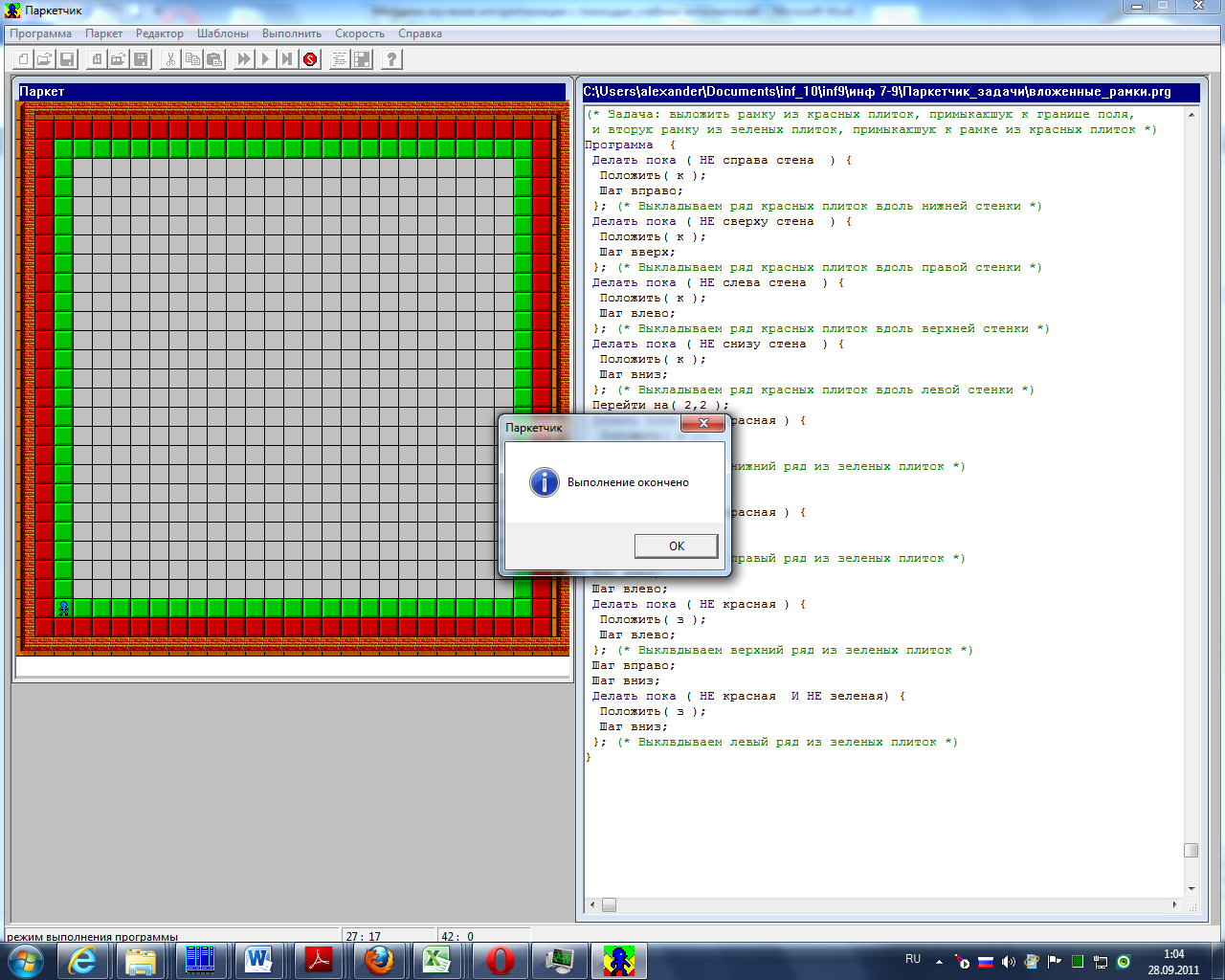


Рис. 1. Поле Паркетчика, программный листок и окно диагностики выполнения программы.

Каждая клетка на поле имеет свой адрес, чтобы Паркетчик знал, где ему предстоит выполнять работу. На экране своего компьютера игровое поле видно как бы сверху, поэтому уместно говорить о горизонтальных и вертикальных рядах клеток. Так, клетка, стоящая в 5-м столбце и 7-й строке, имеет адрес (5, 7). Строки нумеруются снизу вверх и в адресе клетки вначале указываем столбец, а затем строку (фактически это привычная школьникам система координат).

Чтобы выложить тот или иной орнамент, Паркетчик может переходить с любой клетки на соседнюю. Иными словами, для Паркетчика допустимы следующие четыре действия:

• Шаг вверх;

• Шаг вниз;

• Шаг вправо;

• Шаг влево.

Может Паркетчик и сразу прыгнуть на указанную ему клетку поля. Делаетэто он по команде Перейти на (..., …), только вместо многоточий в круглых скобках, конечно, должны быть указаны координаты клетки.

Кроме того, у Паркетчика есть еще два допустимых действия:

• Положить красную плитку;

• Положить зеленую плитку.

Выполняя их, Паркетчик кладет плитку указанного цвета на то поле, на котором он стоит.

Паркетчик имеет также допустимое действие Снять плитку. Надо тут же пояснить, что выполнить это действие Паркетчик может только в том случае, если на клетке, где он находится, лежит плитка какого-то (неважно какого!) цвета.

Осталось сообщить, что в начале игры Паркетчик всегда находится в левом нижнем углу, т. е. в клетке (1, 1).

Для работы с Паркетчиком на теоретических занятиях (при изучении его допустимых действий, алгоритмических конструкций, составления алгоритмов до выхода на компьютерный практикум) удобно иметь макет поля Паркетчика размером 10 на 10, выполненный на (фанерном) планшете, и комплект плиток (как правило, достаточно двух десятков), каждая из которых с одной стороны выкрашена в красный цвет, а с другой в зеленый. В середине каждой клетки на поле имеется отверстие, а в середине каждой плитки вставлен и закреплен небольшой стержень (см. рис. 2). Во время демонстрации плитка с помощью этого стержня закрепляется на поле Паркетчика.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | • | • | • | • | • | • | • | • | • | • |  |  |
| 9 | • | • | • | • | • | • | • | • | • | • |  |  |
| 8 | • | • | • | • | • | • | • | • | • | • |  |  |
| 7 | • | • | • | • | • | • | • | • | • | • |  |  |
| 6 | • | • | • | • | • | • | • | • | • | • |  |  |
| 5 | • | • | • | • | • | • | • | • | • | • |  |  |
| 4 | • | • | • | • | • | • | • | • | • | • |  |  |
| 3 | • | • | • | • | • | • | • | • | • | • |  |  |
| 2 | • | • | • | • | • | • | • | • | • | • |  |  |
| 1 | • | • | • | • | • | • | • | • | • | • |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |  |

а) Планшет с полем для Паркетчика б) Вид плитки с торца

(закрашенные клетки означают,

что на них установлены плитки)

Рис. 2

Иногда поступают наоборот: на поле в середине каждой клетки делают небольшой штырек, а в плитках отверстие. Тогда плитки надеваются на этот штырек. Конечно, в таком виде плитки хранить удобнее, но когда плитка на поле со всех сторон окружена другими, бывает не так-то легко снять ее со штырька.

Впрочем, многие школы имеют сегодня в своем арсенале интерактивную доску. Использовать её гораздо удобнее, если подготовить на ней поле Паркетчика.

На таком планшете или доске можно предлагать учащимся показывать результаты выполнения заданий, которые получаются у них при исполнении алгоритмов «вручную». Мы, к примеру, считаем полезным разобрать на теоретическом занятии примеры семантических ошибок, которые могут иметь место при работе Паркетчика. Для этого можно предложить следующее задание.

|  |
| --- |
| Задание 1. Рассмотрите следующий алгоритм:  Шаг вверх;  Шаг вправо;  Положить (з);  Шаг вправо;  Снять плитку;  Положить (к);  Шаг вверх;  Шаг влево;  Положить (к);  Исполните его для каждого из паркетов, изображенных на рис. 3. Объясните, почему в некоторых ситуациях Паркетчик отказывается работать, выдавая сообщение; «Ошибка во время исполнения». |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

а) б) в)

Рис. 3.

Выполнение этого задания естественно проводить с использованием планшета, о котором говорилось выше. Разумеется, программа может быть другой; другими могут быть и орнаменты на поле Паркетчика.

Отметим, что именно визуализация Паркетчиком исполнения алгоритма позволяет эффективно проверять правильность результата и отыскивать места логических ошибок. Нами не предусмотрен режим пошаговой отладки программ, но имеющаяся возможность менять скорость исполнения программы позволяет без труда отслеживать ход исполнения алгоритма и локализовывать места ошибок. Прекратить исполнение программы можно нажатием клавиши ESC или щелкнув мышкой на иконке с буквой S на верхней панели. Синтаксические ошибки автоматически выявляются во время компиляции, а семантические диагностируются Паркетчиком в ходе исполнения программы. Среди синтаксических ошибок отметим одну, наиболее часто встречающуюся: в команде «Положить (к)» буква «к» набирается в латинском регистре. Будет ли эта буква набрана в верхнем регистре (т.е. заглавной) или нижнем роли не играет.



Рис. 4. Выбор команды из меню Шаблоны

Мы рекомендуем приучать учащихся использовать меню Шаблоны (см. рис. 4) для набора программ это весьма ускоряет набор текста программы и, что более важно, сводит к минимуму появление синтаксических ошибок. Это опять-таки позволяет сосредоточить внимание учащихся именно на алгоритмической сути решаемой ими задачи, а не поиске пропущенной или лишней точки с запятой, открывающейся или закрывающейся скобки.

С той же целью – встраивание в текст программы готовых команд и синтаксических конструкций – можно воспользоваться правой клавишей мыши при расположении её курсора на программном листке (см. рис. 5).

При первоначальной загрузке Паркетчика и программный листок (так мы будем называть окно, в котором записывается программа) и поле Паркетчика автоматически очищаются. Обычно школьниками (особенно на первых порах) по умолчанию предполагается, что на поле Паркетчика нет никаких плиток. Но уже после первого исполнения программы на поле могут остаться какие-то плитки, поэтому при повторном запуске той же или исправленной программы Паркетчик может попытаться положить плитку на клетку, где плитка уже лежит. Однако допустимым действием для Паркетчика является укладка плитки только на свободную клетку, поэтому в подобной ситуации Паркетчик сообщит об ошибке во время исполнения программы. Чтобы избежать этой коллизии, надо перед очередным запуском программы очистить поле Паркетчика, выбрав в верхней строке меню Паркет, а в этом меню пункт Создать новый паркет.

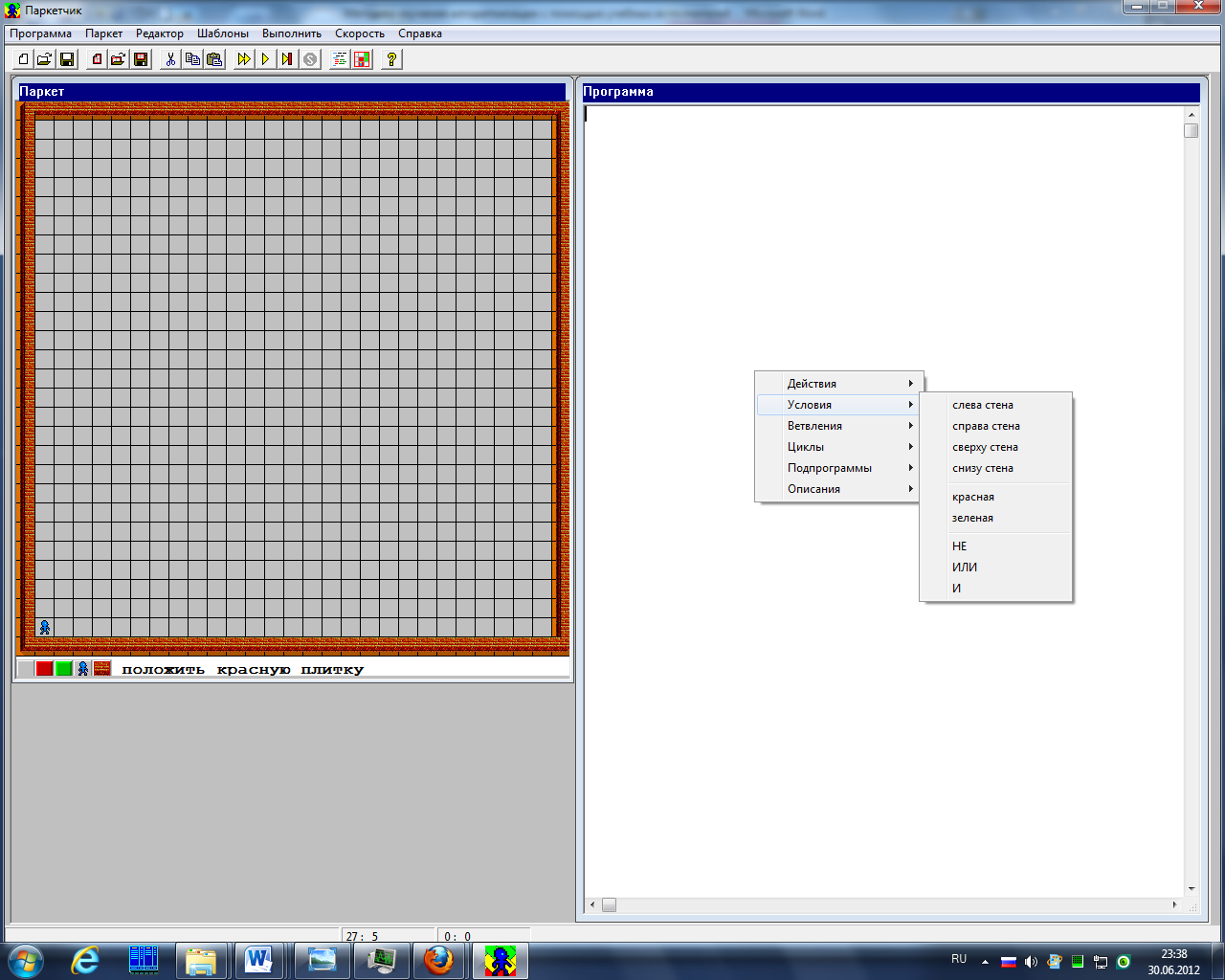


Рис. 5. Выбор условия в контекстном меню

Файл с программой для Паркетчика является фактически обычным текстовым файлом. Его редактирование может, вообще говоря, осуществляться и вне программной среды Паркетчика.

При сохранении программы имя файла автоматически получает расширение prg, при сохранении паркета расширение prk. Поэтому для сохранения программы или паркета в соответствующем окне, которое появляется после того, как нажата клавиша F2 или выбран соответствующий пункт в меню (на рис. 6 показано меню для записи программы), нужно набрать имя без какого бы ни было расширения. Напомним, что выход в верхнюю строку со списком различных меню осуществляется нажатием клавиши F10.

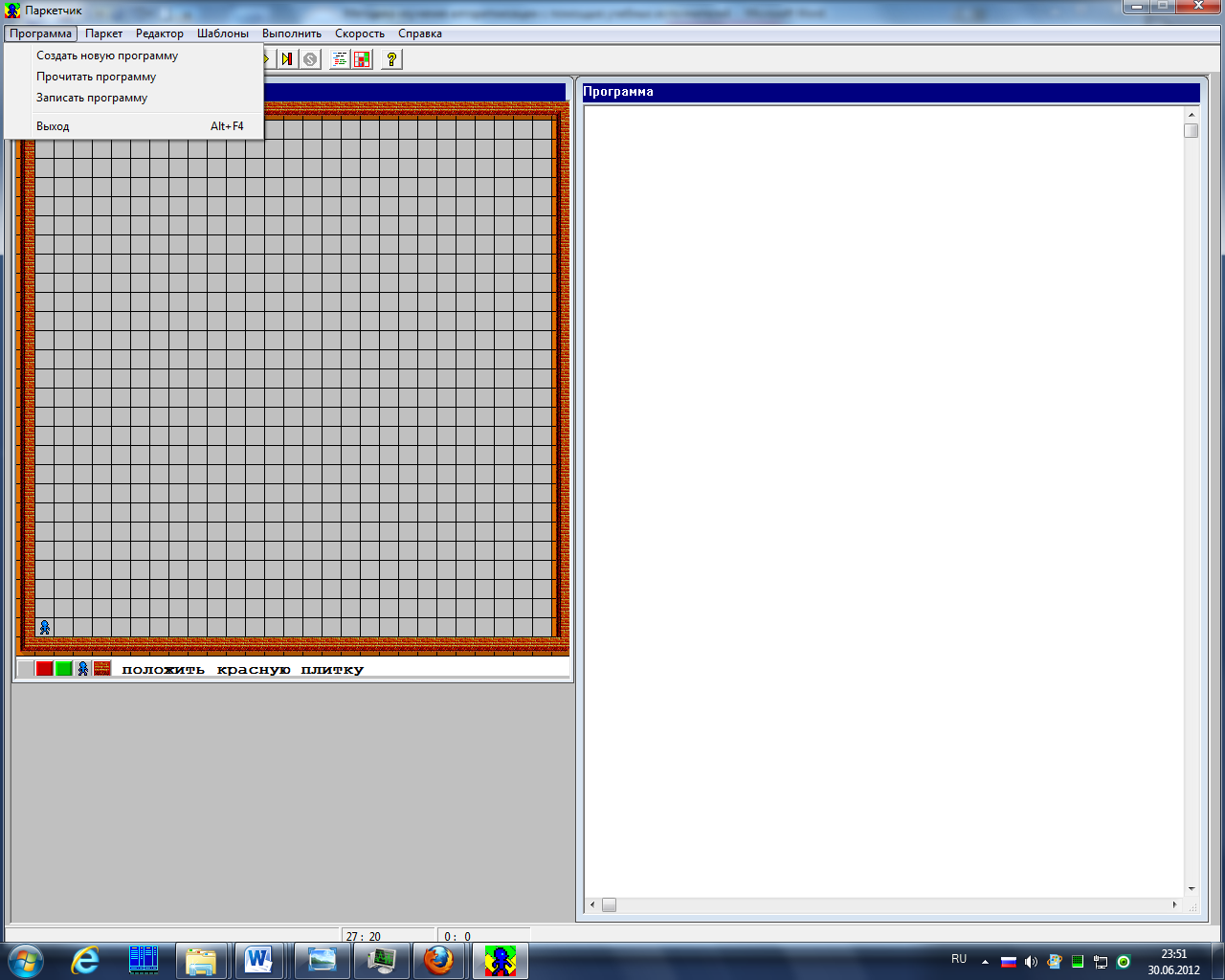


Рис. 6. Выбор пункта меню для сохранения программы

Полезно, на наш взгляд, предложить учащимся войти в меню Скорость и изменить значение скорости, после чего посмотреть, как стала исполняться программа. В дальнейшем умение уменьшить скорость понадобится им для поиска ошибок в программах; для долго работающих программ, наоборот, бывает выгодно увеличить скорость.

Учащиеся могут создавать на поле Паркетчика те или иные орнаменты, так сказать, в ручном режиме. Для этого надо выбрать мышкой нужную плитку, щелкнув на советующем её изображении, расположенном ниже поля Паркетчика, а затем щелкнуть на соответствующей клетке поля или проводя курсором мышки полоску на поле Паркетчика. Чтобы убрать плитку с поля. Надо проделать то же самое, выбрав серый квадратик.

Вот примеры двух заданий, которые позволяют учащимся освоиться с интерфейсом Паркетчика.

Задание 2. На рисунке 7 приведены узоры, используемые в народном творчестве. Составьте алгоритмы для Паркетчика, выполнив которые он выложит эти орнаменты.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

а) Русский б) Удмуртский в) Белорусский

Рис. 7. Национальные народные орнаменты

Задание 3. а) Нарисуйте какой-нибудь небольшой орнамент, приятный вашему глазу. Составьте алгоритм для Паркетчика по выкладыванию этого орнамента.

б) Предложите соседу по парте в роли Паркетчика исполнить составленный вами алгоритм и проверьте, получился ли тот орнамент, который вами был задуман.

**Методика изучения алгоритмических конструкций с помощью исполнителя Паркетчик**

Составление линейных алгоритмов по существу обсуждено нами в предыдущем пункте. Поэтому речь пойдет об освоении учащимися таких конструкций как ветвление, цикл и вспомогательные алгоритмы. В каком порядке изучать эти конструкции – вопрос методических предпочтений учителя (или авторов того учебника, по которому работает данный учитель). Ниже приведена наша аргументация, почему методически целесообразно изучать алгоритмические конструкции в следующей последовательности: циклы в форме «пока», ветвления, вспомогательные алгоритмы, циклы со счетчиком.

Изучение циклов позволяет продемонстрировать учащимся главное преимущество компьютера перед человеком выполнение большого числа действий за короткое время. Ведь даже весьма короткий циклический алгоритм, составить который не так уж долго, при исполнении может потребовать выполнения нескольких сотен действий, с которыми компьютер справится намного быстрее, чем человек. Для алгоритмов с ветвлениями ситуация принципиально иная: в алгоритме требуется записать больше действий, чем может быть на самом деле придется исполнять из двух альтернатив всегда выполняется только одна. Поэтому в случае изучения ветвлений до циклов учащиеся нередко ропщут, что проще “вручную” проверить выполнение условия и указать компьютеру, что делать дальше, чем формализовать требуемое условие и аккуратно расписать обе альтернативы. У школьников создается впечатление, что времени на создание алгоритма с ветвлением уходит больше, чем на его исполнение в режиме пошаговой отдачи команд исполнителю. С циклическими алгоритмами дело обстоит, как было сказано выше, наоборот. Поэтому изучение циклов является мощным мотивирующим фактором к изучению алгоритмов в целом.

Обычно первая программа, которую предлагается составить учащимся, предназначена для выкладывания Паркетчиком одного ряда из красных плиток. Полезно разобрать работу Паркетчика по составленным ими программам на планшете (о нем рассказывалось выше). Нередко учащиеся предлагают следующую программу:

Программа

{Делать пока (не справа стена)

{ Положить (к);

Шаг вправо;

}

}

В этой программе кроется ошибка. Она состоит в том, что на последнюю клетку Паркетчик плитку не положит. Исправляется программа добавлением команды “Положить (к)”, которую надо записать сразу после цикла. Это довольно типичная ошибка в циклических алгоритмах, состоящая в том, что не обрабатывается последний объект, и ее профилактике должно быть уделено должное внимание.

Затем учащимся полезно предложить написать программу выкладывания одноцветной рамки. Простейшее решение состоит в четырехкратном повторении алгоритма выкладывания ряда со сменой направления движения внутри каждого цикла и в условии продолжения цикла. Типичная ошибка: попытка дважды положить плитку на угловое поле.

Вот вариант искомого алгоритма:

## Программа

{

Делать пока (не справа стена)

{ Положить (к);

Шаг вправо;

}

Делать пока (не сверху стена)

{ Положить (к);

Шаг вверх;

}

Делать пока (не слева стена)

{ Положить (к);

Шаг влево;

}

Делать пока (не снизу стена)

{ Положить (к);

Шаг вниз;

}

}

Чтобы замостить плитками все поле, надо организовать уже двойной цикл. И в этом случае полезно предварительно разобрать работу Паркетчика по этой программе на планшете в теоретическом классе.

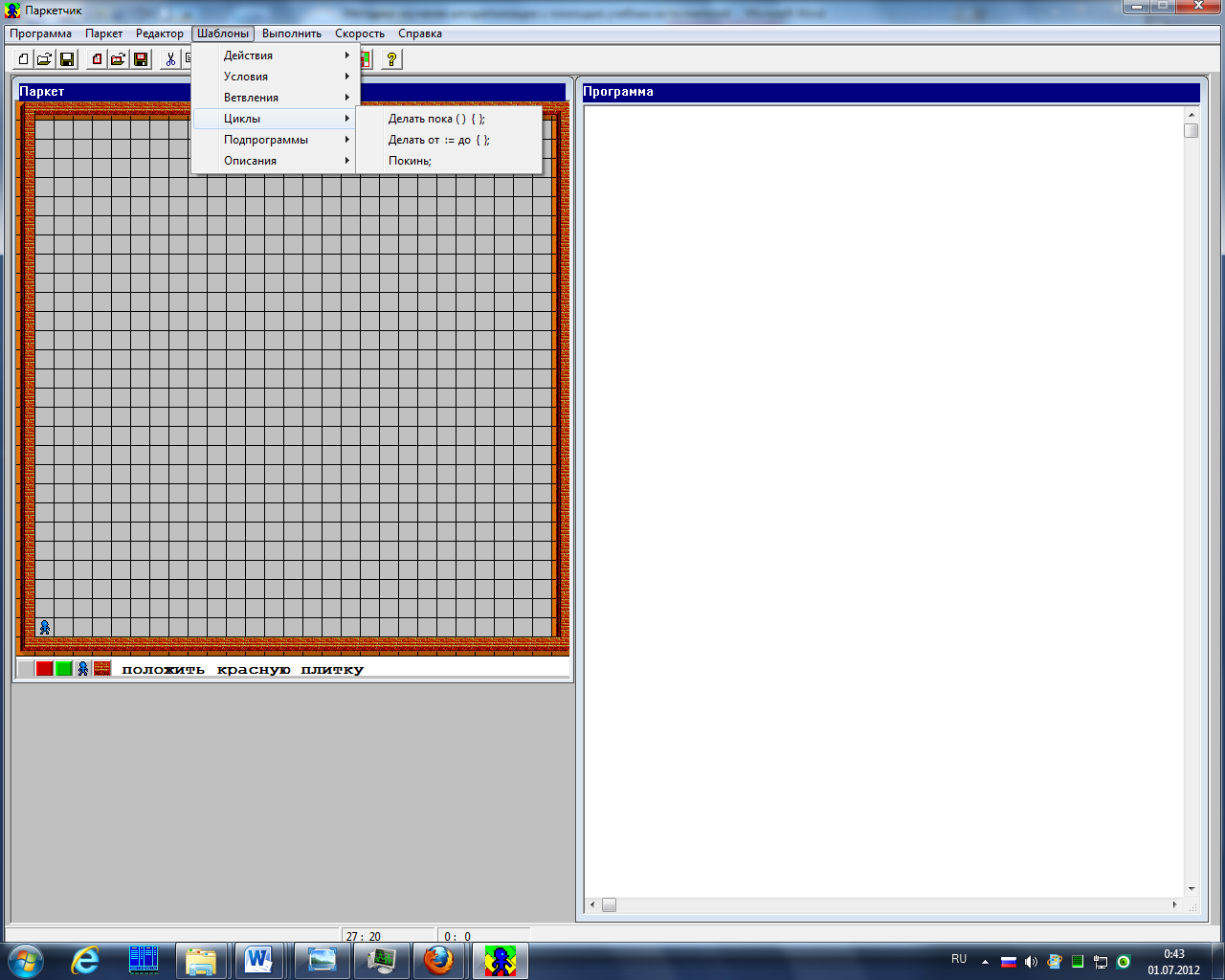


Рис. 8. Подменю различных циклических конструкций

После детального теоретического разбора можно приступать к компьютерному практикуму. Учащимся надо подсказать, что для организации циклов нужно войти в меню Шаблоны (клавиша F4 или щелчок левой клавишей мыши на этом пункте) и воспользоваться подменю Циклы (см. рис. 8) и подменю Условия (см. рис. 5).

Большой мотивационный эффект достигается тогда, когда демонстрируется работа этой (или ей аналогичной) программы до того, как учащиеся начинают составлять свои программы для решения задачи. Знание, что задача разрешима и кто-то с ней справился, пробуждает полезный азарт у учащихся. Демонстрировать можно на интерактивной доске или на рабочих местах учеников, а чтобы учащиеся не видели текст программы, достаточно сжать программный листок.

Вот еще одна задача, которую весьма полезно предложить учащимся.

Задание 4. Составьте алгоритм, выполнив который Паркетчик выложит на поле орнамент из вложенных друг друга рамок чередующихся цветов. Для поля, состоящего из 8 строк и 10 столбцов, такой орнамент приведен на рисунке 9.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Рис. 9

Программа по выкладыванию только двух рамок приведена на рисунке 1. Разберем решение данной задачи.

Самая большая рамка ограничена границами поля и ее построение особого труда не вызывает. Что касается внутренних рамок, то построение каждой следующей ограничивается наличием предыдущей. Значит, Паркетчик должен выкладывать плитки внутренней рамки до тех пор, пока не наткнется на плитки уже выстроенной рамки. Здесь требуется объявить учащимся, что Паркетчик умеет проверять, лежит ли на клетке, где он находится, плитка заданного цвета. Мы, тем не менее, считаем полезным такое опережающее введение этого условия. Ясно и то, что все свои действия по выкладыванию плиток Паркетчик осуществляет до тех пор, пока есть непокрытые клетки (условие продолжения цикла Пока). Приведем возможный вариант программы, решающей эту задачу.

Программа

{ Делать пока (не сверху стена)

{ Положить (з);

Шаг вверх;

}

Делать пока (не справа стена)

{ Положить (з);

Шаг вправо;

}

Делать пока (не снизу стена)

{ Положить (з);

Шаг вниз;

}

Делать пока (не слева стена)

{ Положить (з);

Шаг влево;

} (\* закончили выкладывать рамку вдоль границы поля \*)

Шаг вверх;

Шаг вправо;

Делать пока ((не з) и (не к))

{ Делать пока ((не з) и (не к))

{ Положить (к);

Шаг вверх;

}

Шаг вниз;

Шаг вправо;

Делать пока ((не з) и (не к))

{ Положить (к);

Шаг вправо;

}

Шаг влево;

Снять плитку;

Делать пока ((не к) и (не з))

{ Положить (к);

Шаг вниз;

}

Шаг влево;

Шаг вверх;

Делать пока ((не з) и (не к))

{ Положить (к);

Шаг влево;

}

Шаг вправо;

Шаг вверх;

Делать пока ((не з) и (не к))

{ Положить (з);

Шаг вверх;

}

Шаг вниз;

Шаг вправо;

Делать пока ((не з) и (не к))

{ Положить (з);

Шаг вправо;

}

Шаг влево;

Снять плитку;

Делать пока ((не к) и (не з))

{ Положить (з);

Шаг вниз;

}

Шаг влево;

Шаг вверх;

Делать пока ((не з) и (не к))

{ Положить (з);

Шаг влево;

}

Шаг вправо;

Шаг вверх;

} (\* конец цикла построения внутренних рамок \*)

}

Отметим еще, что в записи условия продолжения цикла построения внутренних рамок мы использовали конструкцию сложного условия. Если учитель не хотел бы использовать сложные условия, здесь можно предложить учащимся использовать эквивалентный ему двойной цикл

Делать пока (не к)

{ Делать пока (не з)

{ …

…

}

}

В предложенной программе дважды встречаются фразы, заключенные в круглые скобки со звездочками. Это комментарии. Они пропускаются транслятором с языка Паркетчика и могут стоять в начале или в конце любого оператора. К примеру, на рисунке 1 на программном листке в форме комментария записано условие задачи.

Еще несколько задач и демонстрационных программ вы найдете в папке с Паркетчиком.

Начать изучение ветвлений полезно с предложения учащимся привести примеры алгоритмов, в которых имеются альтернативы. Важно добиваться, чтобы при этом учащиеся каждый раз четко формулировали проверяемое условие, обеспечивающее выбор одной из альтернатив. Вот несколько простейших примеров, с которых можно начать рассмотрение конструкции ветвления:

Если на улице сухо, то пойти гулять, иначе остаться дома читать книжку.

Если на светофоре зеленый свет, то переходить улицу, иначе оставаться на месте.

Если синоптики обещают дождь, то взять с собой зонт, иначе его оставить дома.

И так далее. Мы убеждены, что и учитель, и ученики легко продолжат список подобных примеров.

Полезно рассмотреть и примеры, когда в качестве второй альтернативы ничего не нужно делать:

Если после набора номера в трубке слышны короткие гудки, то повесть трубку.

Если дождь кончился, то закрыть зонт.

Если на дороге ведется ремонт, то поехать в объезд.

Мы уверены, что и здесь легко можно подобрать немало примеров различных жизненных ситуаций.

Значительное внимание при объяснении конструкции ветвления следует уделить роли операторных скобок. Именно они предотвращают неоднозначность в выделении зоны действия условия. Во многих ранее созданных школьных учебниках по информатике для обозначения такой зоны предлагалось использовать специальный признак «Конец ветвления». В данном учебнике мы сочли целесообразным придерживаться в этом вопросе позиции Н.Вирта в разработанном им языке программирования Паскаль. В этом языке роль открывающей операторной скобки играет служебное слово begin, а закрывающей — служебное слово end. Преимуществ такого подхода несколько.

Во-первых, нет необходимости различать ограничители зоны действия алгоритмической конструкции, т.е. иметь три различных ограничителя — конец ветвления, конец цикла и конец подпрограммы (процедуры).

Во-вторых, такой подход точнее отражает идею структурного программирования, поскольку фрагмент, заключенный в операторные скобки, можно рассматривать как целостный структурный элемент, не зависящий от того, в какой конкретно алгоритмической конструкции он используется.

Отсюда вытекает и третье преимущество данного подхода: при нисходящем проектировании можно рассматривать фрагмент в операторных скобках как независимый элемент, который можно самостоятельно подвергать разработке методом пошаговой детализации.

Вот одно из заданий и его решение.

Задача 5. Вдоль границы поля выложена рамка из плиток. В ней встречаются как красные, так и зеленые плитки. Требуется все красные плитки этой рамки заменить на зеленые.

Программа

{

Делать пока (не справа стена)

{ Если (к) то {Снять плитку; Положить (з)}

Шаг вправо;

}

Делать пока (не сверху стена)

{ Если (к) то {Снять плитку; Положить (з)}

Шаг вверх;

}

Делать пока (не слева стена)

{ Если (к) то {Снять плитку; Положить (з)}

Шаг влево;

}

Делать пока (не снизу стена)

{ Если (к) то {Снять плитку; Положить (з)}

Шаг вниз;

}

}

Еще несколько заданий и программ приведено в папке с Паркетчиком.

Конфигурация поля Паркетчика может изменяться не только за счет варьирования размеров, но и с помощью выставления на клетки поля колонн. Несколько колон образуют стену. Паркетчик не может проходить сквозь стены или становиться на клетку, занятую колонной. Используя этот инструмент, поле Паркетчика можно превращать в довольно сложный лабиринт, например, такой, какой показан на рисунке 10 (зеленой плиткой обозначен выход из лабиринта, она может быть установлена на любой свободной клетке поля). Сильным учащимся можно предложить составить программу для Паркетчика, исполнив которую он найдет выход из любого лабиринта, созданного на его поле, при условии, что лабиринт связен, т.е. каждая внутренняя стена лабиринта соприкасается (возможно «углом») с внешней стеной. Именно такой лабиринт изображен на рисунке 10. Наиболее известный вариант обхода такого лабиринта – это так называемое «правило правой руки»: Паркетчик должен двигаться по свободным клеткам так, чтобы справа от него постоянно была стена (он как бы держится за неё правой рукой, откуда и пошло название этого правила). Разумеется, можно столь же успешно реализовывать «правило левой руки».

Если же лабиринт не связен, как, например, такой, что изображен на рисунке 11, то гарантировать достижение выхода уже нельзя. Начав двигаться из левого нижнего угла по правилу правой руки, Паркетчик, пройдя вдоль внешней стены, вернется в исходную точку и никогда не окажется в центре лабиринта, отмеченного зеленой плиткой. Для данного алгоритма в таком лабиринте его исполнение зациклится.

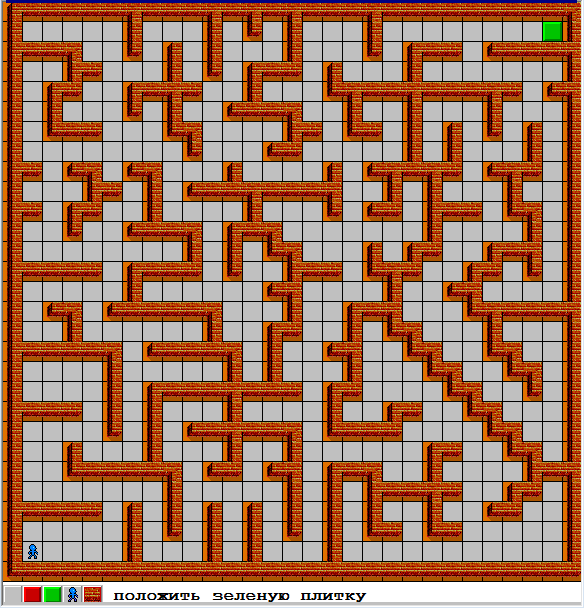


Рис. 10. Лабиринт на поле Паркетчика

Однако, если Паркетчику разрешить как-то помечать клетки, на которых он уже был (например, кладя на них красные плитки), то можно составить алгоритм нахождения выхода из уже произвольного лабиринта. Решение этой довольно непростой задачи может стать основой для проекта, разрабатываемого одним или несколькими учениками.

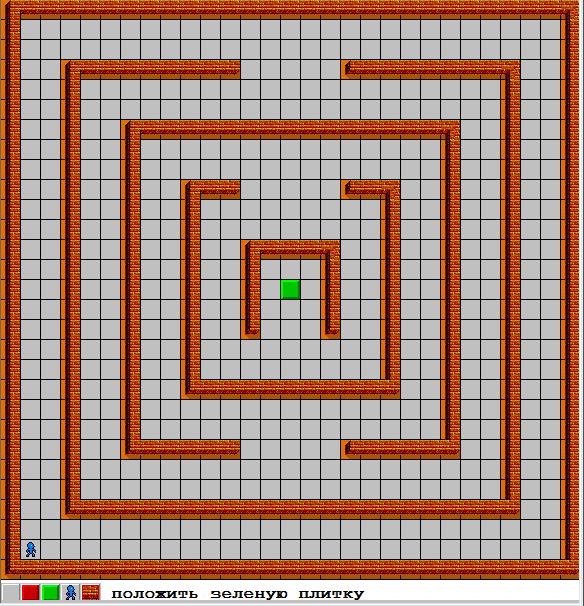


Рис. 11. Несвязный лабиринт

**Методика изучения переменных и структур данных с помощью исполнителя Паркетчик**

Освоение основных алгоритмических конструкций вовсе не связано с обязательным введением понятия переменной и операции присваивания ей значения. Это никак не означает, что мы хотим принизить роль вычислительных алгоритмов в реальной практике. Мы здесь следуем тому методическому принципу, что разнородные понятия легче осваиваются учеником, если они разнесены в курсе, а не собраны в одном месте и изучаются одновременно.

Для Паркетчика реализованы переменные двух типов: целый и цвет. Над переменными целого типа можно выполнять стандартные арифметические операции, переменные типа «цвет» используются для передачи параметров во вспомогательный алгоритм и могут принимать только два значения – «красный» и «зеленый» (в программах для Паркетчика можно писать кратко: «к» и «з»). Как нетрудно понять, это фактически переменные булевого типа. С дидактической точки зрения важно, что учащиеся впервые встречаются с переменными нечислового типа.

Появление переменных позволяет использовать конструкцию цикла со счетчиком. Такая конструкция также реализована в языке Паркетчика (см. рис. 8).

Появление возможности использовать переменные, как правило, встречается учащимися с энтузиазмом – многие алгоритмы, составлявшиеся до этого, значительно упрощаются за счет возможности простого подсчета плиток того или иного цвета. Вот пример задачи, которую можно предложить учащимся для создания у них мотивации к введению переменных.

Задание 6. Вдоль нижней границы поля, начиная с крайней левой клетки, выложена полоска из зеленых плиток. Напишите программу для Паркетчика, выполнив которую, он удвоит длину этой полоски.

Без использования переменных эта задача может быть решена следующим образом.

Программа

{

Делать пока (з)

{ Снять плитку;

Положить (к);

Шаг вправо;

Делать пока ( (з) или (к) )

{ Шаг вправо;

}

Положить (к);

Перейти на (1, 1);

Делать пока (к)

{ Шаг вправо;

}

}

Перейти на (1, 1);

Делать пока (к)

{ Снять плитку;

Положить (з);

Шаг право;

}

}

Красные плитки играют роль сигнальной переменной, показывающей, для каких плиток исходной зеленой полоски уже построено удвоение. Задача еще более усложняется, если потребовать, чтобы использовались только плитки зеленого цвета. Вот возможный вариант решения:

Программа

{

Делать пока (з)

{ Шаг вверх;

Положить (з);

Шаг вправо;

Шаг вниз;

} (\* Создана дублирующая полоска \*)

Перейти на (1, 2);

Если (з) то

{ Снять плитку;

Шаг вниз;

Делать пока (з)

{ Шаг вправо;

}

Положить (з);

}

Перейти на (1, 2);

Делать пока (не (з) и (не справа стена))

{ Шаг вправо;

Если (з) то

{ Снять плитку;

Шаг вниз;

Делать пока (з)

{ Шаг вправо;

}

Положить (з);

Перейти на (1, 2);

}

} (\* Перенос дублирующей полоски в конец исходной \*)

}

В обеих программах предполагается, что в исходной полоске количество плиток не более половины ширины поля.

Естественно желание сосчитать количество зеленых плиток и затем просто добавить к уже имеющейся полоске еще столько же. Вот реализующая эту идею программа:

Программа

{

Делать пока (з)

{ Снять плитку;

Положить (к);

Шаг вправо;

Делать пока ( (з) или (к) )

{ Шаг вправо;

}

Положить (к);

Перейти на (1, 1);

Делать пока (к)

{ Шаг вправо;

}

}

Перейти на (1, 1);

Делать пока (к)

{ Снять плитку;

Положить (з);

Шаг право;

}

}

Красные плитки играют роль сигнальной переменной, показывающей, для каких плиток исходной зеленой полоски уже построено удвоение. Задача еще более усложняется, если потребовать, чтобы использовались только плитки зеленого цвета. Вот возможный вариант решения:

Программа

цел: Счет, N

{ Счет := 0;

Делать пока (з)

{ Счет := Счет + 1

Шаг вправо;

} (\* Подсчитано количество зеленых плиток \*)

Перейти на (Счет, 1)

Делать от N := 1 до Счет;

{ Шаг вправо;

Положить (з);

}

Краткость и ясность программы способна впечатлить любого школьника.

Для решения нам потребовалось две переменных, одна из них названа Счет, другая N. В качестве имен переменных может использоваться любая комбинация букв русского и латинского алфавитов с цифрами или без них, кроме, разумеется, зарезервированных слов языка Паркетчика.

Но для того, чтобы пользоваться переменными, их нужно объявить в начале программы или подпрограммы. Объявление переменных заключается в указании их имени и типа. Сделать это можно опять-таки используя меню Шаблоны (см. рис. 12).

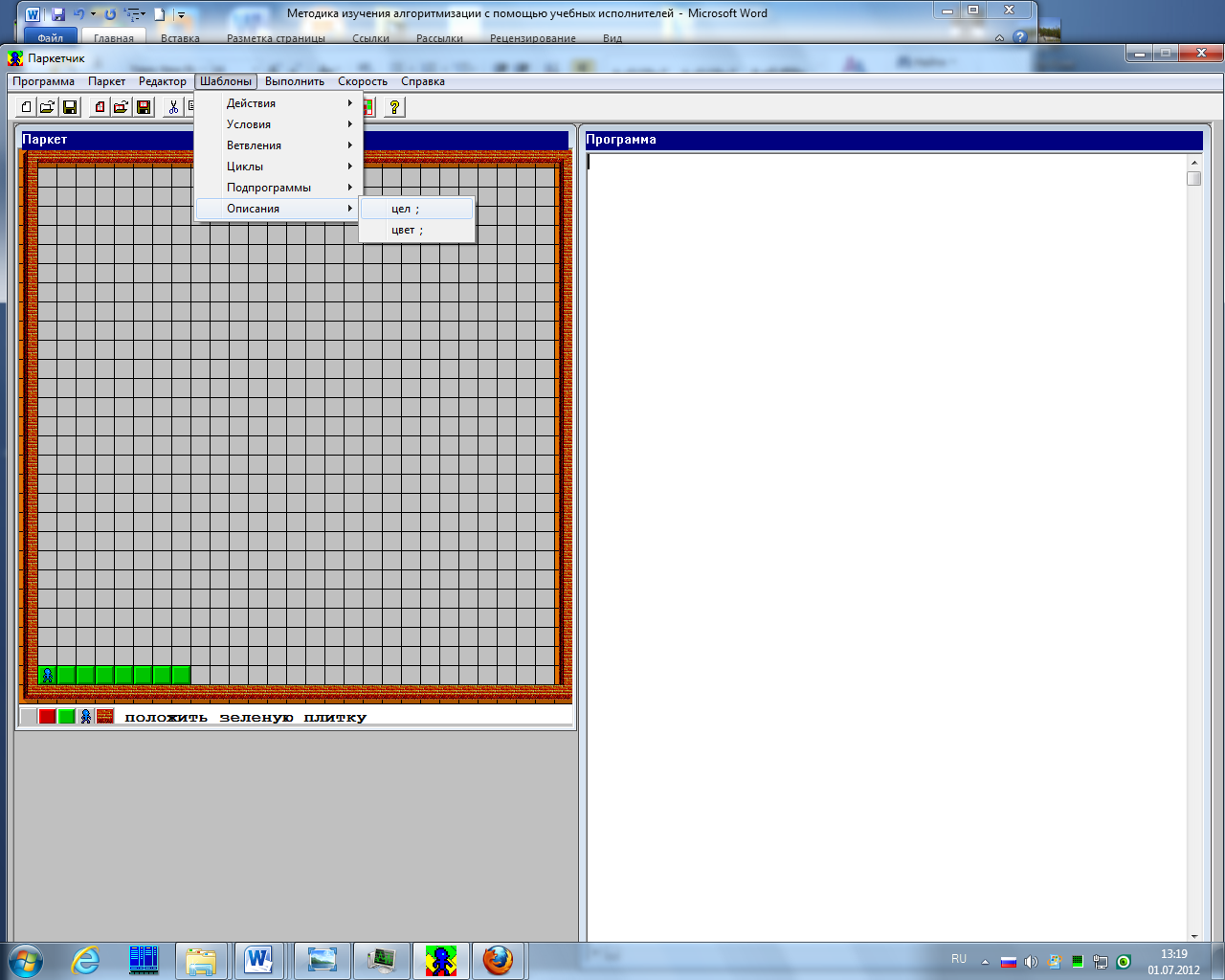


Рис. 12. Подменю описания переменных

Вот еще одна задача, продолжающая данную линию, но решение которой без использования переменных почти невозможно.

Задание 7. Напишите программу для Паркетчика, выполнив которую, он соберет с поля все зеленые плитки и выложит их одна за другой вдоль нижнего края поля (возможно, что при этом он заполнит несколько нижних рядов поля).

На эту задачу можно смотреть как на подготовку к решению задач на сортировку массива подсчетом.

Важным с точки зрения воспитания алгоритмического и вообще продуктивного мышления является освоение механизма вспомогательных алгоритмов и, соответственно, подпрограмм. Это механизм фактически основан на таком методологическом средстве как разбиение задачи на подзадачи, решение которых уже либо известно, либо, по крайней мере, они представляются более простыми для решения. В языке Паркетчика реализован механизм подпрограмм с передачей параметров по значению.

Мотивирующей на изучение механизма вспомогательных алгоритмов (подпрограмм) может быть такая задача.

Задание 8. Напишите программу для Паркетчика, выполнив которую, он создаст на поле орнамент, изображенный на рисунке 13.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 13

Ясно, что написать программу, рисующую одну квадратную рамку, не сложно, но затем повторять её еще семь раз (даже используя режим копирования) совсем не хочется. Создав вспомогательный алгоритм рисования рамки, начиная, например, с её левого нижнего угла, затем будет достаточно только указать координаты той клетки, которая должна стать левым нижним углом новой рамки, и цвет этой рамки. На рисунке 14 показано подменю для оформления подпрограмм и обращения к ним; на программном листке подпрограмма рисования одного квадрата и программа, содержащая два вызова подпрограммы – за одно обращение к подпрограмме рисуется один квадрат заданного цвета в заданном месте.

Разглядывая текст на программном листке, вы видите, что подпрограмма, в отличие от программы, имеет заголовок. После заголовка в скобках идет описание формальных параметров. Оно открывается служебным словом короб, после которого перечисляются имена формальных параметров, являющихся аргументами подпрограммы. После заголовка подпрограммы записывается описание локальных переменных подпрограммы. Обращение к подпрограмме производится по команде Вызвать с указанием имени подпрограммы и в скобках фактических значений передаваемых параметров. Вместо значений здесь могут стоять арифметические выражения от переменных, значения которых вычисляются до момента обращения к подпрограмме. При необходимости можно досрочно прекратить исполнение подпрограммы с помощью оператора Возврат. Мы, однако, рекомендуем приучать учащихся этим оператором не пользоваться (дабы не нарушать парадигму структурного программирования).

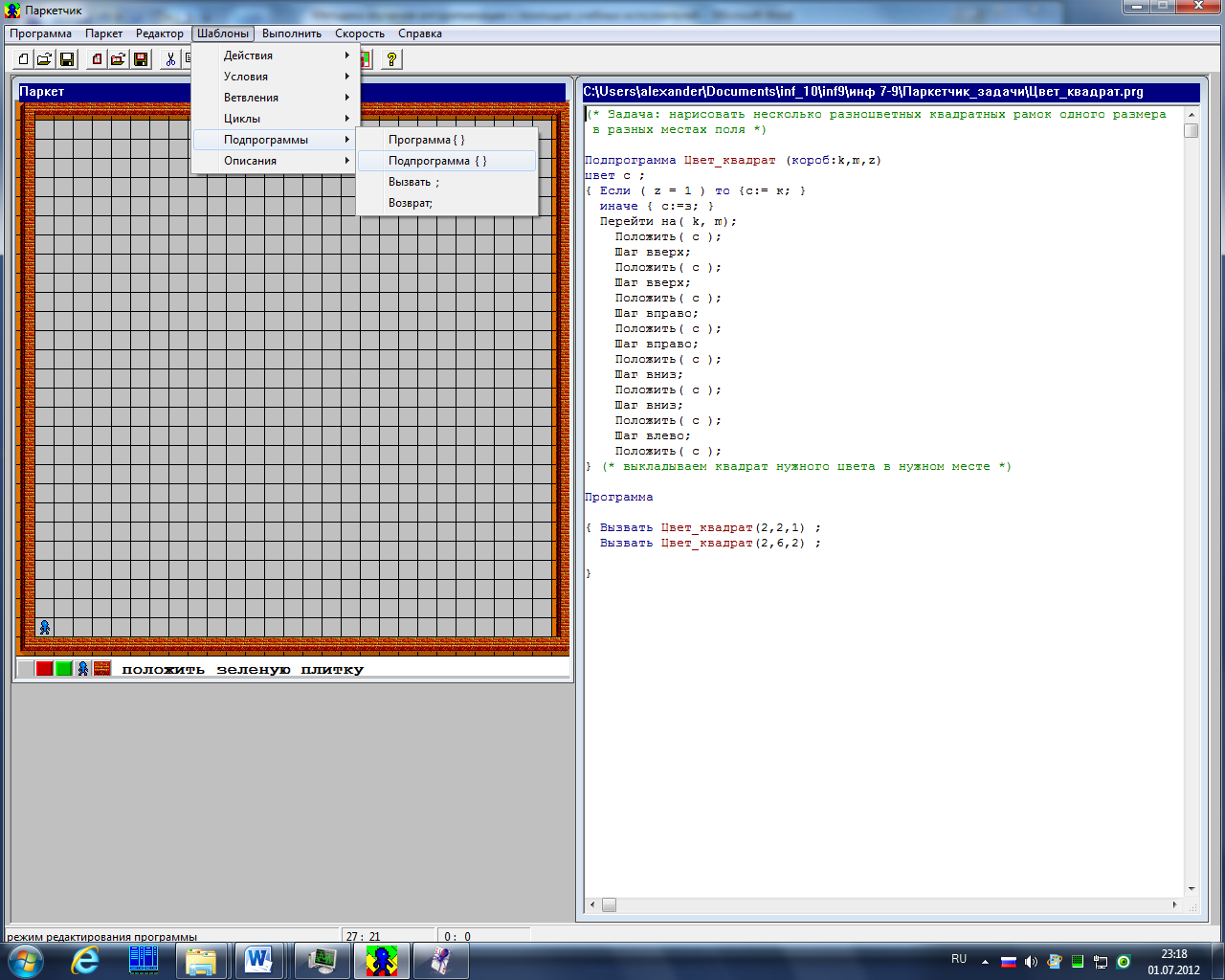


Рис. 14. Оформление подпрограмм в языке Паркетчика

Та версия Паркетчика, на которую опирается это методическое пособие, в реализации механизма подпрограмм имеет ряд ограничений. Во-первых, поскольку реализован однопроходный транслятор с языка Паркетчика, подпрограммы следует размещать до текста основной программы. Во-вторых, все передаваемые параметры – целочислены, это позволило не описывать их ни в основной программе, ни в подпрограмме.

Мы уже отмечали, что решение задач на разработку алгоритмов позволяет формировать и развивать у учащихся не только познавательные универсальные учебные действия, но и действия, относящиеся к регулятивной и коммуникативной группам. Проиллюстрируем это разбором следующего задания.

Задание 9. На рисунке 15 показан орнамент из красных плиток, который изображает слово РОБОТ. Требуется составить программу для Паркетчика, исполнив которую он выложит этот орнамент.

а) Разбейтесь на группы по 3 – 4 человека и обсудите, какие подпрограммы полезно иметь, чтобы выполнить это задание.

б) Распределите между собой, кто какую подпрограмму будет писать, и выберите руководителя вашей рабочей группы, который будет писать основную программу. Согласуйте с ним, как будут передаваться параметры из основной программы в подпрограмму.

в) Напишите нужные программы и подпрограммы.

г) (Минипроект) Требуется разработать комплекс подпрограмм по выкладыванию орнаментов, каждый из которых изображает какую-либо букву русского алфавита. Комплекс должен быть таким, чтобы с его помощью можно было писать программу выкладывания орнамента в виде слова. Для букв М, Ы и Ю вы можете воспользоваться изображениями, приведенными на рисунке 16. Для других букв придумайте изображения самостоятельно и обсудите предложенные варианты в группе. Возможно, вы решите, что не все буквы должны быть одного цвета.

д) Напишите программы для рисования орнаментов, изображающих слова ТОПОТ, ШОРТЫ, КОМИК.

е) Придумайте еще два-три слова, состоящие из пяти – семи букв и напишите для них программы рисования этих слов.

ж) Сравните результаты работы разных групп. (Совет: предварительно сформулируйте критерии, по которым будет происходить сравнение, например, эстетический вид получающихся орнаментов, рациональность и экономичность комплекта подпрограмм и т.п.)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 1.15

Необходимость деловой коммуникации с очевидностью заложена в пунктах а), б), г) и е). Работа учащихся над выполнением пунктов в) и д) могут показать учителю, в какой мере у них сформированы исполнительский и технологический уровни владения компетенциями. При выполнении заданий г) и ж) можно оценивать уровень овладения экспертными компетенциями – ведь решения, которые будут приниматься при выполнении этих пунктов должны опираться на уже сделанную ранее работу. Более того, выполнение пункте г) может демонстрировать учителю, в какой степени учащиеся владеют аналитико-синтезирующими компетенциями – будут ли они перестраивать технологию выполнения заданий с учетом приобретенного опыта при выполнении заданий а) – в).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 16.

Отвлекаясь от собственно вопросов преподавания алгоритмизации, хотим сказать, что подобная схема формирования компетенций и коммуникативных УУД, должна, на наш взгляд, присутствовать при организации всякой проектной деятельности учащихся.

Вернемся к вопросам изучения алгоритмизации. Отметим, что язык Паркетчика допускает использование механизма рекурсии. Ниже приведена довольно простая рекурсивная программа, просто создающая на поле Паркетчика прямоугольник заданной высоты, состоящий из красных плиток.

(\* Задача: выложить прямоугольник из красных плиток ширины 10 и запрашиваемой высоты \*)

Подпрограмма Линия (короб:y)

цел i;

{

Делать от i := 1 до 10 {

Перейти на (i, y);

Положить (к);

}

Если (y > 1) то {

Вызвать Линия(y – 1)

}

}

Программа

цел n ;

{ Запросить n;

Вызвать Линия(n)

}

Сильным учащимся можно предложить запрограммировать для Паркетчика классическую задачу о Ханойских башнях. Поле Паркетчика выглядит для этой задачи так, как показано на рисунке 17. Количество дисков на стержне можно выбрать любым в пределах высоты стержня.

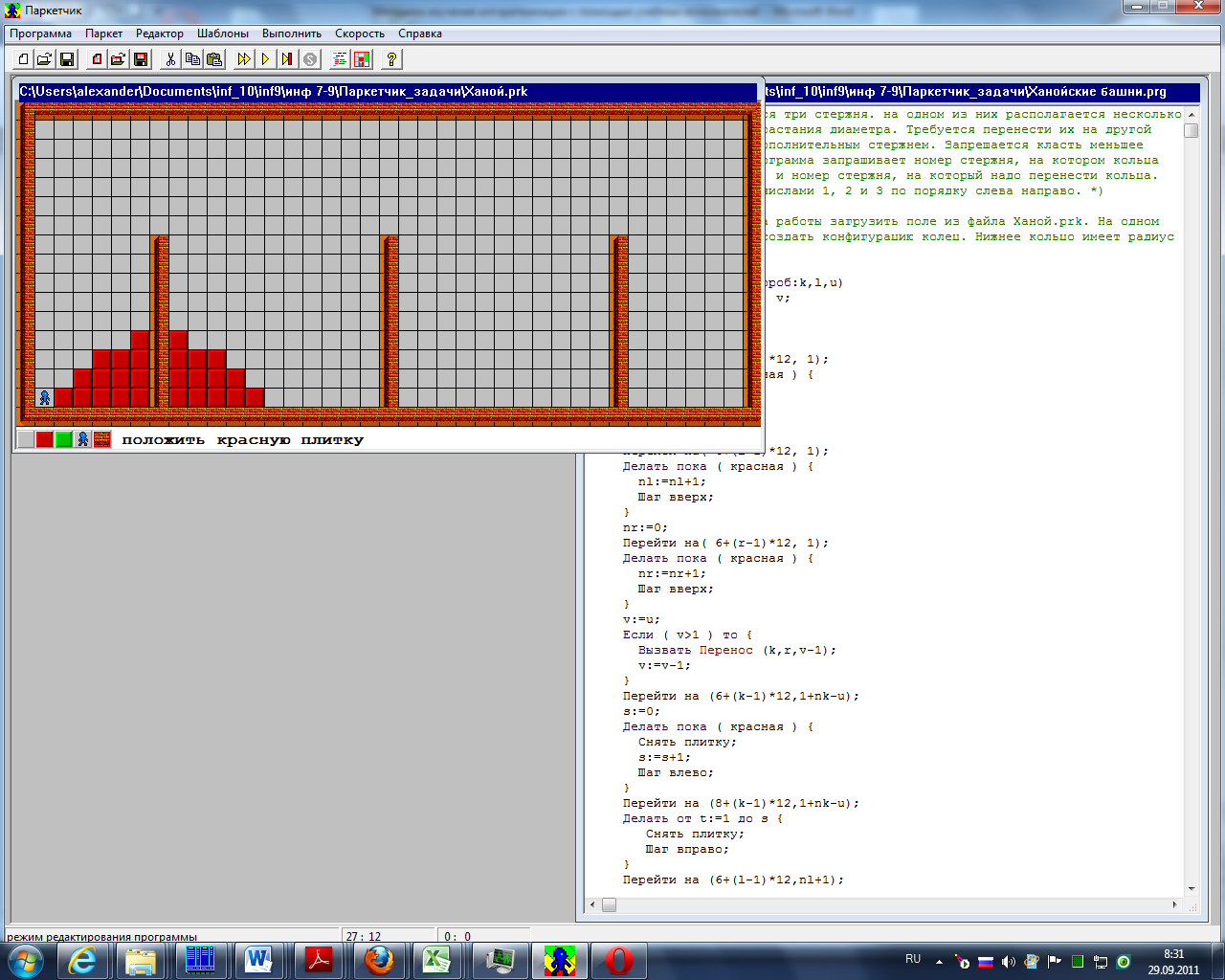


Рис. 17. Поле Паркетчика для задачи «Ханойские башни»

Паркетчик не только способствует эффективному освоению приемов алгоритмизации с использованием переменных, но и знакомству с различными структурами данных. Совершенно ясно, что само поле Паркетчика моделирует структуру данных, которая называется «двумерный массив». Выставляя на поле дополнительные стены, легко смоделировать такие структуры, как стек и дек. Любой лабиринт или орнамент на поле Паркетчика фактически является структурой типа «граф». К примеру, рассмотренная в конце предыдущего пункта задача о поиске выхода из связного лабиринта по правилу правой руки – это ни что иное, как стандартный алгоритм обхода связного графа в глубину.

Вот ещё одна задача, в которой ярко демонстрируется применение алгоритмов на графах.

Задание 10. **Задача одноцветного перехода**.

На поле Паркетчика выложен некоторый орнамент из зеленых плиток (красных плиток на поле нет). Если Паркетчик, не сходя с этого орнамента, может перейти из левого нижнего угла в правый верхний так, что за каждый ход он будет перемещаться только на соседнюю клетку, то нужно все зеленые плитки орнамента заменить на красные; в противном случае оставить все как есть. Составить для Паркетчика соответствующий алгоритм.

Идея решения состоит в следующем. Во-первых, ясно, что если в левом нижнем углу нет зеленой плитки, то делать ничего не нужно. Если же там есть зеленая плитка, то заменим ее на красную. Во-вторых, обходя каждый раз все поле, будем заменять на красные те зеленые плитки, которые соседствуют с уже имеющимися к этому моменту красными. Иными словами, красными плитками отмечаются те клетки поля, которые достижимы Паркетчиком с соблюдением правил, указанных в условии задачи. Поэтому, если в какой-то момент в правом верхнем углу окажется красная плитка, то все оставшиеся на поле зеленые плитки надо заменить на красные. Если же в какой-то момент ни одна зеленая плитка не будет заменена на красную, а в правом верхнем углу все еще лежит зеленая плитка, то добраться до этой клетки Паркетчику никогда не удастся, и, значит, надо вернуть все красные плитки в исходное зеленое состояние. А начать надо, разумеется, с определения размеров поля.

Знатоки программирования увидят в этом решении реализацию так называемого волнового алгоритма. А сама задача — это определение того, принадлежат ли две заданные вершины графа одной компоненте связности.

Но изучение алгоритмов на графах больше относится всё же к олимпиадной тематике, и в общеобразовательном курсе подобные задачи могут предлагаться только сильным учащимся. Что же касается алгоритмов обработки массивов, то это довольно стандартный материал школьного курса информатики. В частности, школьники должны владеть каким-либо алгоритмом сортировки элементов одномерного массива. Обычно задача сортировки формулируется для одномерных массивов, составленных из попарно сравнимых элементов — чисел, символов и т.п. Однако исполнение алгоритмов, обрабатывающих подобные массивы, не визуализировано, и потому их понимание и отладка нередко представляют значительные трудности. Вот как в освоении таких алгоритмов может помочь Паркетчик.

Задание 11. **Сортировка**.

На поле Паркетчика располагается несколько вертикальных полосок из красных плиток, нижняя плитка каждой полоски касается нижнего края поля Паркетчика (столбчатая диаграмма). Напишите программу для Паркетчика, после исполнения которой столбцы расположатся в порядке не возрастания их высоты. Например, исходная позиция, изображенная на рисунке 18 а), должна быть преобразована в ситуацию, представленную на рисунке 18 б)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

а) б)

Рис. 18

Обсуждая это задание со школьниками, можно остановиться на рассмотрении различных алгоритмов сортировки. Наиболее известными являются алгоритм с помощью выбора максимального элемента и алгоритм перестановки соседних элементов (метод «пузырька»). Суть первого из этих алгоритмов состоит в том, что из последовательности элементов выбирается наибольший и он ставится на первое место, затем среди оставшихся тоже выбирается наибольший и он ставится на второе место и т.д. Второй алгоритм состоит в том, что исполнитель, встретив два рядом расположенных элемента, стоящие не в том порядке, меняет их местами. Проход по массиву данных осуществляется столько раз, сколько необходимо для того, чтобы в массиве исчезли пары соседних элементов, стоящих в неправильном порядке. Можно предложить части учеников реализовать первый алгоритм, а другой части — второй. Но возможно, конечно, что учащимися будут придуманы и другие алгоритмы.

🙞 🙜

Многолетняя практика использования данного учебного исполнителя показывает его высокую эффективность при обучении школьников алгоритмизации.

Коснувшись методики обучения алгоритмизации с использованием исполнителя Паркетчик, мы оставили в стороне многие другие вопросы воспитания у учащихся алгоритмического мышления и алгоритмической культуры. С нашими рекомендациями по этим вопросам можно ознакомиться в пособиях [8 – 10].

**Литература**

1. Гейн А.Г., Сенокосов А.И., Юнерман Н.А. Информатика и информационные технологии: учеб. для 8 кл. общеобразоват. учреждений – М: Просвещение, 2009. – 175 с.
2. Гейн А.Г., Юнерман Н.А., Гейн А.А. Информатика: учеб. для 7 кл. общеобразоват. учреждений – М: Просвещение, 2012. – 191 с.
3. Гейн А.Г., Юнерман Н.А., Гейн А.А. Информатика: учеб. для 8 кл. общеобразоват. учреждений – М: Просвещение, 2012.
4. Сенокосов А.И., Гейн А.Г. Информатика: учеб. для 8 – 9 кл. школ с углубл. изуч. информатики – М: Просвещение, 1995. – 255 с.
5. ИТ-кадры 2010. Численность занятых в российской экономике 2009 г. и прогноз потребности 2010-2015 // Электронный ресурс: режим доступа: <http://www.apkit.ru/committees/education/projects/itcadry2010.php>
6. Федеральный государственный образовательный стандарт основного общего образования // Электронный ресурс: режим доступа: <http://standart.edu.ru/catalog.aspx?CatalogId=2588>
7. Фундаментальное ядро содержания общего образования: проект / под ред. В. В. Козлова, А. М. Кондакова. М., Просвещение, 2009. – 48 с.
8. Гейн А.Г. Информатика. Книга для учителя. 7 класс. – М: Просвещение, 2012.
9. Гейн А.Г., Юнерман Н.А. Информатика и информационные технологии: кн. для учителя: метод. рекомендации к учебнику 8 кл. – М: Просвещение, 2005. – 144 с.
10. Гейн А.Г., Сенокосов А.И., Юнерман Н.А. Информатика и информационные технологии: кн. для учителя: метод. рекомендации к учеб. 9 кл. – М: Просвещение, 2008. – 192 с.

1. Есть, конечно, более тонкие классификации программного обеспечения, но для наших целей это не принципиально.

   Данное пособие разработано в рамках исполнения Государственного контракта № 07.Р20.11.0032 от 07.09.2011 г. на выполнение работ (оказание услуг) по подготовке и переподготовке профильных специалистов на базе центров образования и разработок в сфере информационных технологий. [↑](#footnote-ref-1)
2. В контексте данного рассмотрения мы лишь коснулись вопроса, что развитое алгоритмическое мышление (в отличие от умения программировать, понимаемого даже в самом широком смысле как умение составлять и записывать на подходящем языке программирования алгоритмы, управляющие работой компьютера) подразумевает умение человека проектировать и описывать собственную алгоритмизуемую деятельность. На самом деле это один из принципиальных моментов в новых ФГОС. Среди универсальных учебных действий (сокращенно УУД), освоение которых входит в метапредметные результаты обучения, специально выделяется группа регулятивных УУД. В неё включаются (см. [7], с. 40): действия, обеспечивающие организацию учащимся своей учебной деятельности: целеполагание как постановка учебной задачи на основе соотнесения того, что уже известно и усвоено учащимся, и того, что еще неизвестно; планирование — определение последовательности промежуточных целей с учетом конечного результата; составление плана и последовательности действий; прогнозирование — предвосхищение результата и уровня усвоения, его временных характеристик; контроль в форме сличения способа действия и его результата с заданным эталоном с целью обнаружения отклонений и отличий от эталона; коррекция — внесение необходимых дополнений и корректив в план и способ действия в случае расхождения эталона, реального действия и его продукта и т.д. Легко понять, что это «зеркальное» отражение умений по составлению алгоритмов для любого исполнителя, но в данном случае применительно к собственной учебной деятельности. [↑](#footnote-ref-2)
3. Учебный исполнитель Паркетчик является свободно распространяемым продуктом. Его можно скачать с сайта <http://kadm.math.usu.ru/>. Поскольку в настоящее время многие школьники имеют дома компьютер, мы не исключаем, что для выполнения домашних заданий, связанных с составлением программ для Паркетчика, у них возникнет желание иметь его на домашнем компьютере. Мы бы приветствовали передачу им данного программного средства. [↑](#footnote-ref-3)
4. Учебник [1] был разработан под государственный стандарт 2004 г.; учебники [2] и [3] созданы для реализации ФГОС второго поколения. Все учебники имеют гриф «Рекомендовано» МО и Н. [↑](#footnote-ref-4)