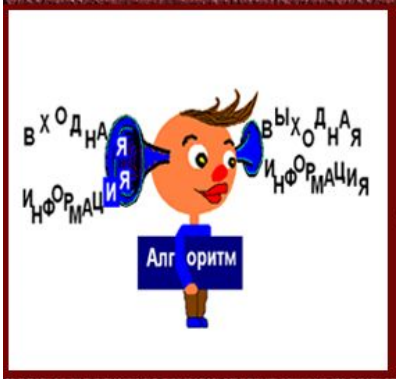
И.В. Николаева Е.П. Давлетярова

Теория и методика обучения информатике

Содержательная линия «Алгоритмизация и программирование»

Учебное пособие



Владимир 2012

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

И.В. Николаева Е.П. Давлетярова

ТЕОРИЯ И МЕТОДИКА ОБУЧЕНИЯ ИНФОРМАТИКЕ

Содержательная линия «Алгоритмизация и программирование»

Учебное пособие

Владимир 2012

УДК 681.142.01 (075)
ББК 32.81.я73
Н 63

Рецензенты:
зав. кафедрой информатизации образования Владимирского института повышения квалификации работников образования
В.А. Власенко

Кандидат физико-математических наук, доцент кафедры специальной техники и информационных технологий Владимирского юридического института Федеральной службы исполнения наказаний России
А.В. Хорошева

Печатается по решению редакционно-издательского совета ВлГУ

Николаева, И.В.

Н 63 Теория и методика обучения информатике. Содержательная линия «Алгоритмизация и программирование»: учеб. пособие / И.В. Николаева, Е.П. Давлетярова; Владим. гос. ун-т. имени Александра Григорьевича и Николая Григорьевича Столетовых. – Владимир: Изд-во ВлГУ, 2012. – 225 с. – ISBN 978-5-9984-0250-0

Учебное пособие предназначено студентам педагогических специальностей, изучающих систематический курс «Теория и методика обучения информатике». В работе сформулированы цели и задачи, которые можно поставить перед учителями и учениками в ходе изучения содержательной линии «Алгоритмизация и программирование», предложена методика изучения выбранных тем данной содержательной линии. Представлены вопросы и задания к семинарским занятиям, лабораторным работам, самостоятельной работе студентов с указанием формы отчета по выполнению заданий.

Пособие будет полезно тем, кто изучает информатику, кто обучает информатике, кто любит информатику, кто еще не знает, что полюбит информатику, – учителям по информатике и информационным технологиям, по математике, учащимся школ.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

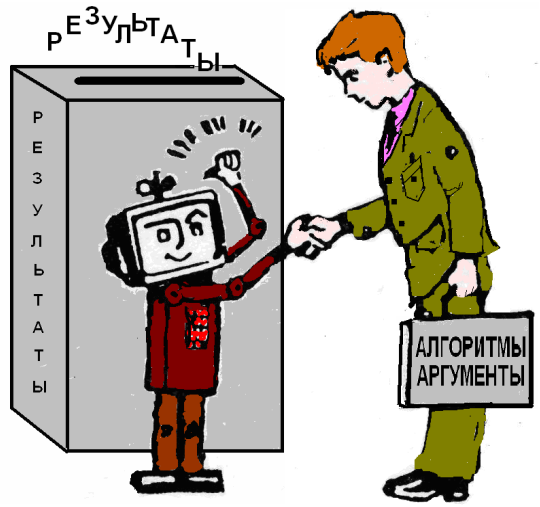
Ил. 246. Табл. 56. Библиогр.: 30 назв.

УДК 681.142.01 (075)
ББК 32.81.я73

ISBN 978-5-9984-0250-0

ОГЛАВЛЕНИЕ

Введение



«… Будущее оставляет мало надежд для тех, кто ожидает, что наши новые электронные роботы создадут для нас мир, в котором мы будем освобождены от необходимости мыслить. Помочь они нам могут, но при условии, что наши честь и разум будут удовлетворять требованиям самой высокой морали»

Н. Винер

Содержательная линия «Алгоритмизация и программирование» является одной из центральных предмета «Информатика и ИКТ». Содержание этой линии определяется через следующие понятия: алгоритм, свойства алгоритмов, исполнитель алгоритма, схема знакомства с исполнителем, схема взаимодействия с исполнителем, способы описания алгоритмов, технологии проектирования алгоритмов, структуры организации данных, управляющие команды организации действий в алгоритмах решения практических задач, основные и вспомогательные алгоритмы, формальное исполнение алгоритмов, способы ручного исполнения алгоритмов, объект, свойства объекта, методы объекта, события, модель, компьютерная модель и т.д.

Изучение алгоритмизации и программирования имеет три целевых аспекта.

А. ***Развивающий аспект***, под которым понимается развитие алгоритмического и логического мышлений обучаемых.

Формирование алгоритмического стиля мышления – это социальный заказ информационного общества образовательным учреждениям. Информатика имеет в своем составе систему понятий, которая позволяет в полном объеме сформировать у обучаемого умения и навыки, имеющие общекультурную, общеобразовательную, общечеловеческую ценность, которые нужны каждому человеку в современном информационном обществе; информатика может предложить, столь

необходимый для формирования алгоритмического мышления, дидактический инструментарий.

Перечислим основные умения и навыки, соответствующие алгоритмическому стилю мышления, которые формирует информатика: умения и навыки планирования структуры действий, необходимых для достижения цели при помощи фиксированного набора средств; структурирование сообщений; понимание и использование формальных способов кодирования решения задачи; технические навыки и умения взаимодействия с компьютером; проектирование и построение информационных и компьютерных моделей; инструментирование всех видов деятельности; умение производить структурный анализ задачи, разбивать большие задачи на малые, сводить нерешенные задачи к решенным.

В процессе составления алгоритмов решения задач, а затем преобразования (кодирования) алгоритмов в программы для определенных исполнителей, у учащихся развиваются навыки проведения логических рассуждений и характерные для дедуктивного мышления умения находить логические следствия из заданных начальных условий, способности абстрагировать, т.е. выделять в конкретной ситуации существенные свойства объекта, процесса, отвлекаясь от несущественных, умения анализировать, сравнивать, обобщать, ставить вопросы, давать четкие ответы, выдвигать конструктивные решения, обосновывать предложенные решения, специализировать, определять понятия, составлять суждения. Все это формирует мышление учащихся и способствует развитию их речи, особенно таких качеств выражения мысли, как порядок, точность, ясность, краткость, обоснованность. Каждое из перечисленных умений и навыков имеет самостоятельное и очень важное значение в системе навыков умственных действий, необходимых любому современному человеку.

В. ***Практический аспект.*** Важной целью изучения предмета «Информатика и ИКТ» является получение учениками опыта построения и исследования моделей реальных объектов на компьютере, в частности, с использованием систем программирования. Фундамен-

том любых компьютерных технологий являются компьютерные модели, которые отображают реальные объекты, явления, человеческую деятельность в компьютерные информационные структуры; именно в своей алгоритмической части информатика имеет дело с формальным описанием внешнего мира.

С. ***Программистский аспект.*** Профессия программист является достаточно распространенной и престижной, изучение программирования в курсе информатики позволяет обучаемому испытать свои способности к такого рода деятельности. Научно-технический прогресс, успехи образования, развитие культуры в значительной мере определяются уровнем компьютеризации и наличием программного обеспечения для соответствующих направлений. Следовательно, необходимы кадры, способные создавать компьютерную технику и программное обеспечение, отвечающие современным требованиям. Чем раньше начать готовить эти кадры, тем весомее будут задачи, которые они смогут решить за свою жизнь. Формирование «программистских» навыков и умений в позднем возрасте связано с ломкой сложившихся представлений и стиля мышления, что существенно осложняет процесс формирования. Изучение элементов программирования в предмете «Информатика и ИКТ» общеобразовательных учреждений не вызывает сомнений. Углубленное изучение приемов и методов современного программирования необходимо в школах с техническим и физико-математическим профилями.

Какие же «программистские» навыки мышления должен приобретать обучаемый в школьном курсе информатики?

Предмет «Информатика и ИКТ» должен знакомить учащихся с идеями и методами как процедурного, так и непроцедурного программирования, вырабатывать соответствующие типы мышления. Изучать идеи и методы программирования лучше с использованием тех систем, которые специально созданы для обучения искусству программирования (рис. 1).

Краткая структурная схема изучения темы «Алгоритмизация и программирование»

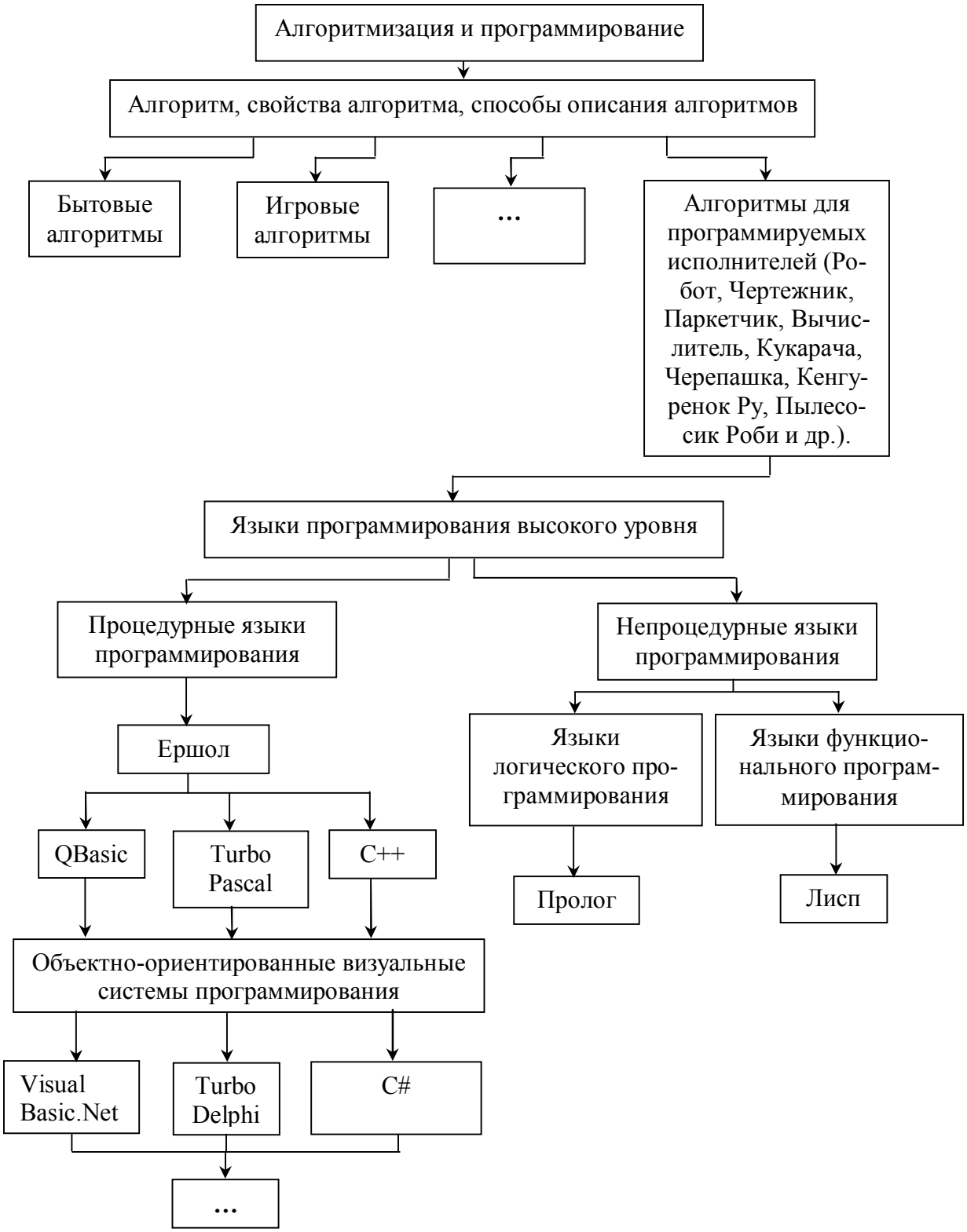


Рис.1

Требования к знаниям и умениям учащихся при изучении содержательной линии «Алгоритмизация и программирование»

Учащиеся должны знать:

- что такое алгоритм; какова роль алгоритма в системах управления;
- основные свойства алгоритма;
- способы записи алгоритмов: словесный, операторный, графический (блок-схемы), на алгоритмических языках программирования;
- схему знакомства с исполнителем;
- схемы взаимодействия с исполнителями;
- основные управляющие команды организации действий в алгоритмах: следование, ветвление, циклы; структуры алгоритмов;
- назначение вспомогательных алгоритмов; технологии построения сложных алгоритмов: «сверху вниз» или метод последовательной детализации алгоритма, «снизу вверх» или использование готовых вспомогательных алгоритмов;
- способы организации (типы) данных;
- назначение систем программирования;
- этапы исполнения компьютером программы, написанной на языке программирования высокого уровня;
- содержание этапов разработки программы на языке высокого уровня: алгоритмизация – кодирование – отладка – тестирование;
- организацию действий и организацию данных в написании алгоритмов на одном из универсальных языков программирования.

Учащиеся должны уметь:

- приводить примеры алгоритмов, определять свойства выбранного алгоритма;
- определять исполнителя; приводить примеры задач для выбранного исполнителя; писать алгоритмы решения задач для определенного исполнителя;
- пользоваться частично формализованным языком для описания алгоритмов, языком блок-схем;
- анализировать структуру алгоритма;

- определять по выбранному методу решения задачи, какие алгоритмические конструкции могут войти в алгоритм;
- выделять подзадачи; определять и использовать вспомогательные алгоритмы; пользоваться различными методами проектирования алгоритмов: «сверху вниз», «снизу вверх» и др.;
- писать алгоритмы (линейные, разветвляющиеся, циклические) на алгоритмических языках высокого уровня.

Рекомендации по изучению языков программирования

Процедурное программирование

В языках процедурного программирования основным методом программирования является разбиение решения задачи на дискретные шаги и их последовательное описание в программе (алгоритме) на соответствующем языке, т.е. написание программы (алгоритма).

Язык начального обучения процедурному программированию должен отражать в своей структуре все основные концепции современного программирования, он должен содержать хорошо развитый аппарат ветвления, циклов, процедур, достаточно богатые структуры данных. Изучение этого языка должно позволить обучаемому в дальнейшем легко перейти к программированию на любом другом процедурном языке высокого уровня.

Для знакомства с процедурным программированием можно рекомендовать школьный алгоритмический язык (Ершол или Е-язык).

Ершол реализован на всех персональных компьютерах, используемых в общеобразовательных учреждениях. Учитывая насыщенность программы по информатике и информационным технологиям, на базовом этапе обучения информатике можно ограничиться изучением и использованием для решения задач только языка Ершол [13, 1].

Располагая достаточным количеством часов, можно показать учащимся как изученные идеи и методы процедурного программирования могут быть реализованы с использованием другого языка программирования (Visual Basic.net, Turbo Delphi, Visual C# и др.). При изучении второго языка программирования способы формирования

предыдущих понятий должны служить примером для создания моделей изучения новых понятий. Результативным приемом обучения, в этом случае, является использование параллельного описания базовых управляющих команд организации действий, данных, программ на изученном и изучаемом языках программирования.

Краткая история языков программирования Ершол, Turbo Pascal, QBasic

Язык программирования Ершол

Школьный алгоритмический язык (Е-язык, Ершол) разработан в СССР в середине 80-х годов XX века (1985-1988 гг.) для обучения школьников и студентов современным методам программирования. В 1985 г. в соответствии с постановлением Совета министров РСФСР от 22 мая «О мерах по обеспечению компьютерной грамотности учащихся средних учебных заведений и широкого внедрения ЭВТ в учебный процесс» с 1 сентября 1985 г. в 9 классах всех средних общеобразовательных школ впервые было введено преподавание нового предмета «Основы информатики и вычислительной техники». На подготовку для преподавания этого предмета были направлены опытные учителя математики и физики. Центром подготовки учителей информатики в нашей области стала кафедра информатики Владимирского педагогического института и кабинет информатики Института усовершенствования учителей. Занятия на курсах по подготовке учителей информатики, выездные семинары, консультации проводили преподаватели кафедры информатики: Дектярев Исраил Муневич, Москвитина Инга Иосифовна, Николаева Ирина Васильевна; методист кабинета информатики ИУУ Исаев Михаил Ефимович и др.

К 1 сентября 1985 г. были изданы первые учебные пособия для учащихся 9 класса и методические рекомендации для учителей общеобразовательных учреждений по основам информатики и вычислительной техники. Их основной содержательной линией была линия «Алгоритмизация и программирование». Для кодирования алгоритмов в учебных пособиях использовался школьный алгоритмический язык (Е-язык, Ершол), алгоподобная нотация для записи алгоритмов. Его автором считают советского математика, академика АН СССР Андрея Петровича Ершова (1931-1988). С 1985 по 1986 гг. этот язык дорабатывался, и описание ядра окончательной версии появилось в учебном пособии 1988 г. Первые трансляторы языка Ершол были разработаны сотрудниками ЛВТ МГУ под руководством А.Г. Кушниренко и А.В. Михалева в 1985 г. Первый вариант программного обеспечения, поддерживающего курс «Основы информатики и вычислительной техники», был разработан для машин серии DEC, СМ ЭВМ и Искра-226. Впоследствии программное обеспечение было разработано для всего ряда вычислительной техники, поставляемой в школу, в том числе: Корвет, Яма-

ха, УКНЦ, Агат, IBM PC и др. Разработанное программное обеспечение получило высокую оценку ВНТК «Школа» (временный коллектив по созданию педагогических программных продуктов для общеобразовательных школ и разработке системы опережающего образования на базе новой информационной технологии под руководством академика Е.П. Велихова).

В 1988 г. академик А.П. Ершов в соавторстве с А.Г. Кушниренко и Г.В. Лебедевым написал новый учебник по информатике «Основы информатики и вычислительной техники». В этом учебном пособии подробно изложен синтаксис языка Ершол, рассмотрены примеры его использования для решения практических задач. Позднее язык Ершол был включен в систему «Кумир» – программное обеспечение, поддерживающее курс информатики, разработанное сотрудниками ЛВМ МГУ А.Г. Леоновым и М.Г. Эпиктетовым. Сотрудники ЛВМ МГУ под руководством А.Г. Леонова продолжают дальнейшее совершенствование системы программирования Ершол, адаптируя ее для существующих операционных систем, в частности, для Windows, включая в систему современные методы программирования. Разработки программного обеспечения 2000-2005 гг. соответствуют мировым стандартам и помогают современным школьникам и студентам быстрее освоить элементы программирования, используя дистанционное обучение через Интернет.

Ершол – прекрасный язык для первоначального изучения структурного программирования. Ершол выгодно отличает следующее:

1. Наглядность – во время ввода или исправления программы транслятор Ершола постоянно обрабатывает вносимые человеком изменения, и выдает на полях программы предупреждения о замеченных ошибках или несоответствиях; отслеживаются все синтаксические ошибки, которые, в принципе, обнаружимы при редактировании. Ершол отслеживает также все ошибки, возникающие при выполнении программы – использование неопределенных переменных, выход индекса за границу массива, переполнение и т.д. Отладчик Ершола в пошаговом режиме показывает на полях результаты присваиваний и порядок проверки условий.

2. Объектно-ориентированный подход – конструкция «исполнитель» поддерживает понятие информационной модели и одновременно современную объектно-ориентированную технологию.

3. Открытость – динамическое подключение внешних исполнителей дает возможность преподавателю выбирать те из них, которые он сочтет необходимыми в данном курсе или на данном уроке.

Язык программирования Turbo Pascal

Язык программирования Pascal был разработан профессором Цюрихского Федерального технологического института Никлаусом Виртом для изучения искусства и техники структурного программирования. Предварительное описание

языка появилось в 1968 году. Вирт назвал его в честь великого французского математика, изобретателя и философа XVII века Блеза Паскаля. Первый транслятор языка заработал в 1970 году, а в 1979 году их было уже более 80. В 1982 году был создан международный стандарт Pascal (ISO Pascal Standard). Описание стандартного языка Pascal на уровне пособия появилось в 1987 году.

Turbo Pascal был разработан в 1984 году. Слово «Turbo» в английской лексике обозначает ускорение. Turbo Pascal – это интегрированная система программирования, содержащая редактор текста, транслятор, справочную систему, небольшую операционную систему, отладчик и другие программы.

В 1992 году корпорация Borland выпустила два пакета программирования, основанные на использовании языка Pascal – Borland Pascal 7.0 и Turbo Pascal 7.0 для MS DOS. Для операционной системы Windows в 1991 году был создан Turbo Pascal for Windows, а в 1992 – Borland Pascal with Objects 7.0. В 1995 году была разработана первая версия Delphi. Затем появились Delphi 2.0, Delphi 3.0, Turbo Delphi и т.д., которые продолжают серию Паскаль-ориентированных средств программирования и являются удобным инструментом визуального объектно-ориентированного программирования для Windows.

Язык программирования QBasic

Язык программирования Basic разработан в 1965 г. сотрудниками Дартмутского колледжа под руководством Джона Кемени и Томаса Курца по заказу фирмы General Electric для решения вычислительных задач пользователями, которые не являются профессиональными программистами, в режиме диалога человека с ЭВМ. Название языка происходит от сокращения английских слов Beginner's All – purpose Symbolic Instruction Code (BASIC), в переводе означающих «многоцелевой язык символических инструкций для начинающих». Первой реализацией языка Basic в СССР стала Basic-система для ЭВМ типа БЭСМ-4, М-220, М-222, разработанная в 1970-1971 гг. группой сотрудников Горьковского (ныне г. Нижний Новгород) университета под руководством Ю.Л. Кеткова. В 1972 г. в Вычислительном центре СОАН СССР разработана система Basic-6 для ЭВМ БЭСМ-6, предназначенная для одновременного обслуживания 10 пользователей. Трансляторы языка Basic входили в состав математического обеспечения ЭВМ, выпускаемых в странах членах СЭВ: М6000, СМ-2, СМ-4, ЕСЭВМ.

В середине 1975 г. фирма MITS представила два варианта интерпретатора языка Basic, созданных Биллом Гейтсом и Полом Алленом, основателями и руководителями фирмы Microsoft. Это были первые версии языка Basic, созданные для микроЭВМ. В 1981 г. для ПК фирмы IBM фирма Microsoft разработала несколько версий интерпретатора языка Basic. В 1985 г. фирма Microsoft представила систему программирования Quick Basic (быстрый Бейсик), работавшую в ОС MS-DOS и состоящую из интерпретатора компилирующего типа с развитыми возможностями по созданию и отладке программ и компилятора для генера-

ции эффективного кода. Благодаря усилиям и финансовой поддержке Microsoft язык Basic продолжает активно развиваться. Для операционной системы MS-DOS созданы версии QBasic и Visual Basic for MS-DOS; для операционной системы Windows выходят новые версии Visual Basic; во все офисные приложения компания Microsoft встроила язык Visual Basic for Applications (VBA), используемый в них в качестве своеобразного макроязыка. В наши дни язык Visual Basic.NET является одним из простых и в то же время мощных языков программирования, он используется для создания Windows-приложений любого типа.

Визуальное объектно-ориентированное программирование

В настоящее время при изучении содержательной линии «Алгоритмизация и программирование» предмета «Информатика и ИКТ» общеобразовательных учреждений используют языки визуального объектно-ориентированного программирования Visual Basic.Net, Turbo Delphi, C#, J#. Визуальное объектно-ориентированное программирование является развитием технологии алгоритмического структурного программирования.

Системы визуального объектно-ориентированного программирования являются визуальными, т.к. используют визуальный метод создания графического интерфейса, и объектно-ориентированными, т.к. используют объектный метод построения программного кода.

Объект в объектно-ориентированном программировании является основным понятием. Объект – это совокупность взаимосвязанных полей и методов, существующих как единое целое.

Объектно-ориентированное программирование – это методология программирования, которая основана на представлении программы в виде совокупности объектов. Процесс разработки программы в среде визуального объектно-ориентированного программирования сводится к выбору набора объектов и их свойств, заданию событий и процедур их обработки, которые в совокупности обеспечивают решение поставленной задачи. Объектно-ориентированное программирование характеризуется тремя основными свойствами: инкапсуляцией, наследованием и полиморфизмом.

Инкапсуляция – это объединение в единое целое данных и алгоритмов обработки этих данных. В рамках объектно-ориентированного программирования данные называются полями, а алгоритмы – объектными методами.

Наследование – это свойство объектов порождать своих потомков. Объект-потомок автоматически наследует от родителя все поля и методы. Программист может дополнять объекты-потомки новыми полями и методами.

Полиморфизм (с греческого языка «много форм») – это возможность родственных объектов решать схожие по смыслу проблемы разными способами, изменяя алгоритм того или иного метода в потомках объекта. Для изменения метода необходимо перекрыть его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющих разную алгоритмическую основу и, следовательно, придающие объектам разные свойства.

Языки логического и функционального программирования

Для знакомства с идеями и методами ***логического программирования*** можно рекомендовать язык Пролог (Prolog). Язык Пролог может рассматриваться в школьном курсе информатики как средство построения несложных баз знаний.

Моделирование знаний – тема искусственного интеллекта, разработка которой в базовом курсе информатики носит пока еще поисковый характер. В будущем в школьном курсе информатики линия искусственного интеллекта, безусловно, получит дальнейшее развитие.

Язык Пролог – это logic programming language (язык логического программирования) или programming in logic (программирование в терминах логики). Язык Пролог описывает не процедуру решения задачи, а логическую модель предметной области задачи – некоторые факты (свойства) относительно объектов предметной области и отношения между этими свойствами, а также правила вывода новых

свойств и отношений из уже заданных. Пролог разработан в начале 70-х годов 20-го века во Франции в Марсельском университете группой специалистов во главе с Алэном Колмероэ. Появление языка связано с работами в области создания искусственного интеллекта. Ценность Пролога в возможности использования его как сложного и тонкого инструмента для разработки высокоспециализированных систем искусственного интеллекта.

Для знакомства с идеями и методами ***функционального программирования*** можно рекомендовать изучение языка Лисп (Lisp). В 1962 году группа специалистов под руководством Дж. Маккарти в Массачусетском технологическом институте разработала язык Лисп. Лисп – это list processing – обработка списков. Программа и обрабатываемые данные в языке Лисп представляются в одной и той же форме, в форме списков. Основными методами функционального программирования являются композиция и рекурсия. Язык Лисп представляет собой реализацию идей теории рекурсивных функций. Язык Лисп – главенствующий язык для создания высокоспециализированных систем искусственного интеллекта в США, язык Пролог – в Японии.

Глава 1. Алгоритмизация

Алгоритмизация – это раздел информатики, изучающий алгоритмы и их свойства, формальные способы записи алгоритмов, методы их проектирования (поиск методов решения задач, реализацию решения в алгоритмы для выбранных исполнителей), доказательство свойств рассматриваемого алгоритма.

С другой стороны, под алгоритмизацией будем понимать сам процесс построения алгоритма решения задачи: выделение этапов процесса обработки данных, определение порядка их выполнения, формальную запись содержания этих этапов.

1. Алгоритм и его свойства

1.1. Понятие алгоритма

Понятие «алгоритм» является одним из основных понятий информатики и математики. Название алгоритм происходит от Algorithmi – латинской формы написания имени великого учёного средневекового Востока Мухаммеда ибн Мусса Аль Хорезми. Его годы жизни приблизительно с 783 по 850 гг. Он сформулировал правила выполнения арифметических действий над многозначными числами.

Что же будем понимать под алгоритмом?

Задача 1. Даны два натуральных числа[1] a и b. Напишите алгоритм нахождения НОД(a, b) – наибольшего общего делителя a и b.

[1] Числа 1, 2, 3, … , которые является результатом счета предметов, будем называть натуральными.

Решение. a и b – входные данные задачи. По существу, мы имеем не одну задачу, а бесконечное множество однотипных задач, столько, сколько существует различных пар натуральных чисел (a; b). Возникает вопрос: существует ли общий метод, позволяющий для любой частной задачи, получаемой подстановкой вместо a и b конкретной пары натуральных чисел, за конечное число шагов определить НОД(a, b).

Для решения задачи необходимо иметь последовательность каких-то действий над входными данными и результатами предыдущих действий, если такие уже имеются, позволяющих найти НОД(a, b). Желательно, чтобы набор этих действий был невелик и в то же время достаточен для нахождения наибольшего общего делителя для любых a, $b \in \mathrm{N}$. Тогда должно существовать нечто такое, обозначим его «А», что после каждого действия оно должно указывать, что делать дальше: либо окончить процесс и указать результат, либо перейти к выполнению следующего определенного действия.

Вот это «А» и принято называть алгоритмом.

Для решения поставленной задачи выберем бездумного исполнителя, в систему команд которого входят команды: найди частное от деления двух натуральных чисел, найди остаток от деления двух натуральных чисел, сравни два натуральных числа и выбери большее или меньшее из них, присвой значения переменным.

Алгоритмом решения поставленной задачи для выбранного исполнителя является, например, алгоритм Евклида.

Шаг 1. Сравни два числа a и b. Обозначь большее число через r, а меньшее через r_1. Найди остаток r_2 от деления r на r_1. Сравни r_2 с нулем, если r_2 равно нулю, то НОД(a, b) присвой значение r_1 и иди на шаг 3, иначе иди на шаг 2.

Шаг 2. Переменной r присвой значение r_1, а r_1 – значение r_2. Найди остаток от деления r на r_1 и обозначь его r_2. Сравни r_2 с нулем, если r_2 равно нулю, то НОД(a, b) присвой значение r_1 и иди на шаг 3, иначе иди на шаг 2.

Шаг 3. Процесс вычисления НОД(a, b) закончи.

Возникает вопрос: всегда ли описанный процесс заканчивается, т.е. всегда ли за конечное число шагов получим нулевой остаток? Легко доказать, что процесс последовательного деления конечен. Любой из остатков является неотрицательным целым числом, и последовательность остатков монотонно убывает. Следовательно, последовательность конечна и последний остаток равен нулю.

Под алгоритмом будем понимать понятное и точное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи.[2]

1.2 Исполнитель. Схема знакомства с исполнителем

Каждый алгоритм строится в расчете на некоторого исполнителя. Для того, чтобы исполнитель мог решить задачу по заданной последовательности команд, необходимо, чтобы он был в состоянии выполнить каждое действие, предписываемое этой последовательностью команд.

Основными характеристиками исполнителя являются: среда, элементарные действия, система команд, отказы.

Рис. 2

Исполнители бывают формальные (бездумные) и неформальные.

Формальный исполнитель имеет жестко заданный список выполняемых команд и при вызове каждой из них непременно совершает элементарное действие. (Если исполнение невозможно, возникает «отказ».) Неформальный исполнитель может и не выполнить

---

[2] *Понятия алгоритма*, которые формируют авторы учебников по «Информатике и ИКТ»: *Алгоритм* – это информационная модель, описывающая процесс преобразования объекта из начального состояния в конечное, в форме последовательности понятных исполнителю команд. *Алгоритм* – это строго детерминированная последовательность действий, описывающая процесс преобразования объекта из начального состояния в конечное, в форме последовательности понятных исполнителю команд. [Н.Д. Угринович]

Алгоритм – это предназначенное для конкретного исполнителя точное описание последовательности действий, направленных на решение поставленной задачи. Можно сказать, что алгоритм – модель деятельности исполнителя алгоритмов. [Л.Л. Босова]

Алгоритм – описание последовательности действий (план), исполнение которых приводит к решению поставленной задачи за конечное число шагов. [Н.В. Макарова]

поданную команду. Примеры неформальных исполнителей: человек, собака. Примеры формальных исполнителей: Робот, Чертежник, Вычислитель, Кенгуренок Ру, Пылесосик Роби [проект «Пилотные школы»], Черепашка [система ЛогоМиры] и др.

Среда – это «место обитания» исполнителя.

Система команд. Каждый исполнитель может выполнять команды из строго заданного списка – системы команд исполнителя (СКИ). Для каждой команды из СКИ должны быть заданы условия применимости и описаны результаты выполнения команды. После вызова команды исполнитель совершает соответствующее элементарное действие.

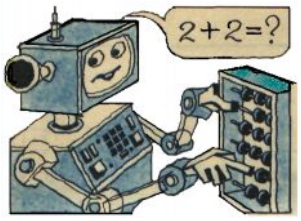
«*Отказы*» исполнителя возникают при вызове команды в недопустимом для данной команды состоянии среды.

Формальное исполнение алгоритма. Исполнитель ничего не знает о цели алгоритма. Он выполняет все полученные команды, не задавая вопросов «почему?» и «зачем?». Когда алгоритм составлен, его исполнение можно поручить автоматическому устройству, например, компьютеру.

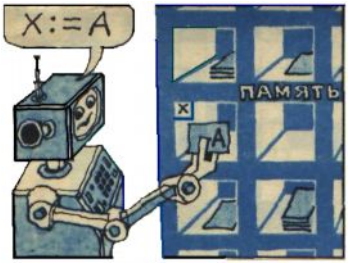
Формальных (бездумных) исполнителей разделим на два класса: *простые* исполнители и *универсальные* исполнители. Простые исполнители не знают порядка выполнения команд алгоритма и умеют выполнять только простые команды: «вверх», «вниз», «закрасить», «право 60», «присвоить переменной *c* значение *a+b*» и другие. К простым исполнителям относятся, например: Робот, Чертежник, Кенгуренок Ру, Пылесосик Роби. Универсальные исполнители знают порядок выполнения команд алгоритма, умеют выполнять составные команды (команды ветвления, команды выбора, команды повторения), умеют обращаться к вспомогательному алгоритму. Простые исполнители – это «инструменты» для реализации алгоритма. А универсальные исполнители – это сочетание «устройства управления» и «инструмента». Роль устройства управления при выполнении алгоритма простыми исполнителями выполняет либо человек, либо какое-то автоматическое устройство. Например, в проекте «Пилотные школы»

Кенгуренком Ру управляет мальчик Кристофер, а Пылесосиком Роби – девочка Милли; в учебном пособии [13] Роботом и Чертежником управляет компьютер, в учебном пособии [1] рассматриваются две схемы управления этими исполнителями – управление человеком и компьютером.

В систему команд универсального исполнителя должны входить, например, такие команды:



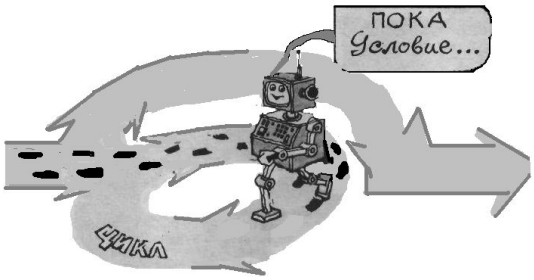
1. *Считай*. Исполнитель должен не только быстро и безошибочно производить все четыре арифметических действия, но и понимать порядок их выполнения, «понимать скобки», вычислять значения переменных величин.

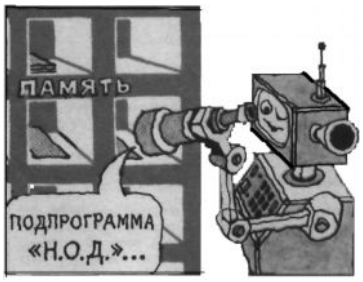


2. *Запомни*. Исполнитель умеет заносить сообщаемые ему значения данных в определенные (известные ему) места своей памяти.

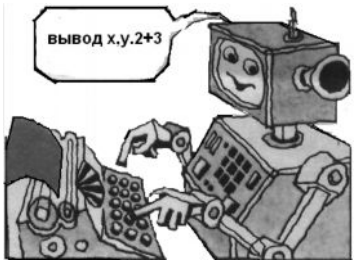
3. *Выбери*. Исполнитель должен уметь сам принимать решения о своих дальнейших действиях в зависимости от обстановки, он проверяет, выполнено ли некоторое сообщаемое ему условие, и если оно выполнено, то он исполняет одну серию команд, иначе – другую серию.

4. *Повтори*. Исполнитель повторяет (быстро и, быть может, многократно) одну и ту же серию команд (цикл), например, пока выполнено определенное условие (условие цикла). Это условие он должен проверить перед каждым выполнением серии, а обнаружив, что оно нарушается, он должен перейти к следующей за циклом команде. Хороший исполнитель, например, современный компьютер, силен именно умением с колоссальной скоростью таким образом работать в цикле.





5. *Вспомни*. Среди информации, записанной в память исполнителя, могут быть данные разных типов (числовые, литерные, графические, звуковые и др.), которые он должен уметь при необходимости



вспомнить. В частности, в память могут быть занесены вспомогательные алгоритмы.

6. *Напечатай*. Исполнитель печатает результаты выполнения программы, постепенно образовавшиеся в нужных местах его памяти в процессе работы.

1.3. Свойства алгоритмов

Алгоритмы обладают следующими свойствами.

Конечность. Исполнение алгоритма всегда заканчивается за конечное число шагов (хотя оно может быть весьма большим).

Дискретность. Выполнение алгоритма разбивается на последовательность законченных действий – шагов. Каждое действие должно быть закончено исполнителем, прежде чем он перейдет к выполнению следующего действия. Произвести каждое отдельное действие исполнителю предписывает специальное указание в записи алгоритма, называемое командой.

Понятность. Каждый алгоритм строится в расчете на конкретного исполнителя, поэтому каждая команда алгоритма должна входить в систему команд этого исполнителя.

Детерминированность. Действия, выполняемые на каждом шаге, однозначно определены входными данными и результатами предыдущих шагов.

Массовость. Алгоритм применим к некоторому классу задач.

Результативность. Исполнение алгоритма всегда приводит от исходных данных задачи к результату, который является решением задачи. В качестве одного из возможных решений может выступать установка того факта, что задача решения не имеет.

2. Способы записи алгоритмов

Существует несколько способов записи алгоритмов, каждый из которых имеет свои особенности: словесный, графический в виде блок-схем, в виде программы на языках программирования и др.

2.1. Словесный способ записи алгоритмов

При словесном способе записи алгоритма пользуются средствами обычного языка, математическими символами и выражениями. Последовательность действий часто оформляется в виде шагов,

которые нумеруются. Достоинством этого способа является наличие возможности записать алгоритм с любой степенью детализации.

Задача 2. Волк, коза и капуста. Исполнитель – *Крестьянин.* На

берегу реки стоит крестьянин с лодкой, а рядом с ним – волк, коза и капуста. Крестьянин должен переправиться сам и перевезти волка, козу и капусту на другой берег. Однако в лодку, кроме крестьянина, помещается либо только волк, либо только коза, либо только капуста. Оставлять же волка с козой или козу с капустой без присмотра нельзя – волк может съесть козу, а коза – капусту. Напишите алгоритм переправы крестьянина, волка, козы и капусты с левого берега на правый.

Решение. Выберем для исполнения составляемого алгоритма исполнителя Крестьянина. Дадим строгое описание исполнителя: где работает Крестьянин, и какие команды он может исполнять. Крестьянин работает на переправе и в его распоряжении находится лодка. Опишем команды, которые научим выполнять Крестьянина: вправо коза; вправо волк; вправо капуста; влево коза; влево волк; влево капуста; вправо ничего; влево ничего. Рассмотрим как выполняет команду «вправо коза» Крестьянин. При выполнении этой команды Крестьянин совершит следующие действия: погрузит в лодку козу, перевезёт её на правый берег, выгрузит её там и остановится в ожидании команды. Аналогично выполняются другие команды.

Алгоритм:

вправо коза; влево ничего; вправо волк; влево коза;	вправо капуста; влево ничего; вправо коза

Задача 3. Переливание воды. Исполнитель – *Водолей*. На столе

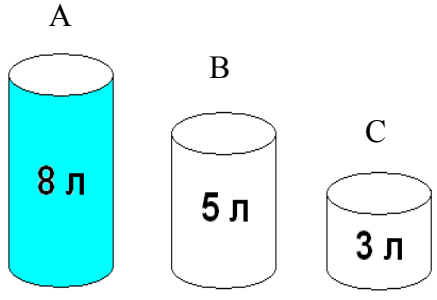


Рис. 3

стоят три ёмкости воды разных размеров. В первую ёмкость вмещается 8 литров воды, во вторую – 5 литров воды, в третью – 3 литра воды. Первая ёмкость заполнена доверху водой, две другие ёмкости пустые.

Напишите алгоритм получения в двух ёмкостях по 4 литра воды, чтобы затем залить воду по 4 литра в два аквариума; нельзя проливать ни капли воды на стол.

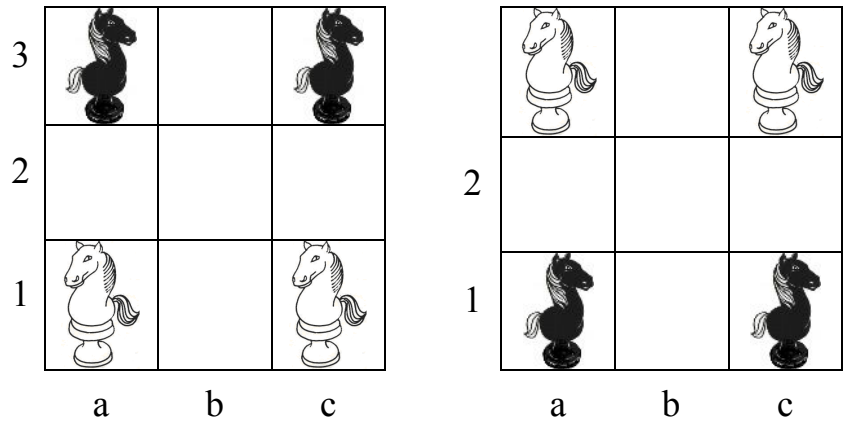
Решение. Выберем для исполнения составляемого алгоритма исполнителя Водолея. Дадим строгое описание Водолея: с чем работает Водолей, какие команды входят в его систему команд (СКИ). Водолей работает с тремя ёмкостями (8 л, 5 л, 3 л), обозначим ёмкости буквами A, B, C. Опишем команды, которые научим выполнять Водолея: перелей из A в B, перелей из A в C, перелей из B в A, перелей из B в C, перелей из C в A, перелей из C в B. Рассмотрим команду «перелей из A в B». Результат этой команды зависит от того, достаточно ли в ёмкости B места для всей воды из ёмкости A. Если да, то ёмкость A становится пустой, а в B оказывается столько воды, сколько было в A и B вместе до переливания. Если же места в B недостаточно, то B становится полной, а в A остается столько воды, сколько не поместилось в B. Аналогично выполняются Водолеем другие команды СКИ.

Таблица-протокол исполнения алгоритма

Алгоритм		Номер шага	Команда	Количество воды в емкостях		
				A	B	C
1 шаг	Перелей из A в B	0		8	0	0
2 шаг	Перелей из B в C	1	Перелей из A в B	3	5	0
3 шаг	Перелей из C в A	2	Перелей из B в C	3	2	3
4 шаг	Перелей из B в C	3	Перелей из C в A	6	2	0
5 шаг	Перелей из A в B	4	Перелей из B в C	6	0	2
6 шаг	Перелей из B в C	5	Перелей из A в B	1	5	2
7 шаг	Перелей из C в A	6	Перелей из B в C	1	4	3
		7	Перелей из C в A	4	4	0

Задача 4. Задача о перестановке четырех коней. Исполнитель – *Конюх*. Напишите алгоритм (для исполнителя Конюха) перестановки коней из начального положения в конечное (рис.4).

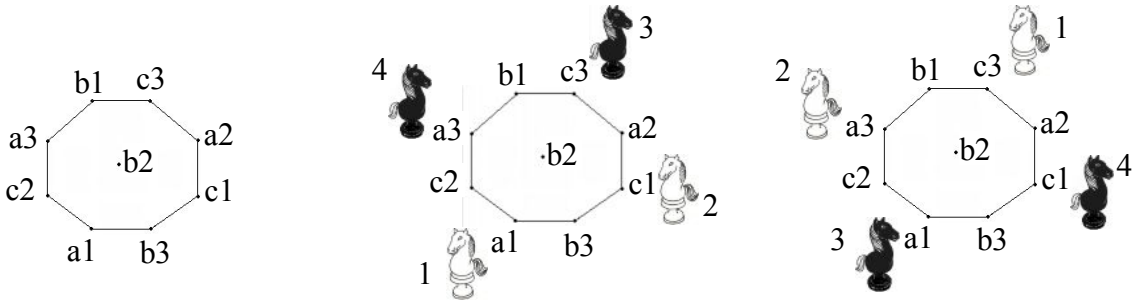
Конь перемещается за один ход либо на два поля по вертикали и одно поле по горизонтали, либо на два поля по горизонтали и одно по вертикали (буквой «Г»). Конь может перепрыгивать через стоящие на его пути другие фигуры, но может перемещаться (ходить) в нашей задаче только на свободные поля.



Начальное положение Конечное положение

Рис. 4

Решение. Каждой клетке доски сопоставим точку на плоскости, и если из одной клетки можно попасть в другую ходом коня, то соединим соответствующие точки линией, получим граф (рис. 5).



Начальное положение Конечное положение

Рис.5

Начальная и конечная расстановки коней изображены на рисунке 5.

Написание алгоритма перестановки коней для исполнителя Конюха становится элементарной задачей.

Алгоритм перестановки коней

Действие, производимое исполнителем, – это одно перемещение какого-нибудь коня на доступную клетку поля. Запись действий формализуем: сначала обозначим поле, с которого надо переставить коня, потом нарисуем стрелку, а затем обозначим поле, на которое надо поставить коня. Один из возможных алгоритмов имеет вид:

Шаги	Номер коня	Команда	Шаги	Номер коня	Команда
1 шаг	1	a1→c2	9 шаг	1	a3→b1
2 шаг	2	c1→b3	10 шаг	2	a1→c2
3 шаг	3	c3→a2	11 шаг	3	c1→b3
4 шаг	4	a3→b1	12 шаг	4	c3→a2

5 шаг	1	c2→a3	13 шаг	1	b1→c3
6 шаг	2	b3→a1	14 шаг	2	c2→a3
7 шаг	3	a2→c1	15 шаг	3	b3→a1
8 шаг	4	b1→c3	16 шаг	4	a2→c1

Задача 5. Легкая пластинка. Исполнитель – *Эксперт*. Имеется 9 пластинок и двухчашечные весы без гирь. Одна из пластинок легче других, но по виду они одинаковые. Как с помощью двух взвешиваний найти более легкую пластинку? Напишите алгоритм определения легкой пластинки.

Решение. Определим исполнителя Эксперта. Эксперт работает на рабочем столе. На столе 9 пластинок, весы и две коробки (склад1, склад2). Эксперт умеет взвешивать пластинки, определять какая из чашек весов с пластинками легче, определять количество предметов, лежащих на столе, делить количество предметов на три равные группы, сравнивать количество предметов.

Алгоритм

1 шаг. Раздели все пластинки на рабочем столе на 3 равные группы.

2 шаг. Положи две группы пластинок на весы, а третью в склад1.

3 шаг. Сравни чаши весов.

4 шаг. Если правая чаша весов легче, то положи на рабочий стол пластинки из правой чаши, а пластинки из левой чаши и из склада1 положи в склад2 и иди на шаг 7, иначе иди на шаг 5.

5 шаг. Если левая чаша весов легче, то положи на рабочий стол пластинки из левой чаши, а пластинки из правой чаши и из склада1 положи в склад2 и иди на шаг 7, иначе иди на шаг 6.

6 шаг. Положи пластинки из левой чаши и правой в склад2 (чаши уравновешены), на рабочий стол положи пластинки из склада1 и иди на шаг 7.

7 шаг. Сосчитай пластинки на рабочем столе.

8 шаг. Если на столе более одной пластинки иди на шаг 1, иначе на шаг 9.

9 шаг. Закончи алгоритм определения легкой пластинки (она лежит на столе).

Задача 6. Деление. Исполнитель – *Вычислитель*. Составьте алгоритм вычисления остатка r от деления m на n, где m и n – натуральные числа.

Решение. Перечислим команды системы команд исполнителя: сравни два натуральных числа, найди разность двух натуральных чисел, присвой значение переменной.

Исходные данные – натуральные числа m и n.

Алгоритм

Шаг 1. Сравни m с n: если $m<n$, то перейди к шагу 2, в противном случае перейди к шагу 3.

Шаг 2. Прими r равным m. Перейди к шагу 4.

Шаг 3. Замени значение m значением $m-n$. Перейди к шагу 1.

Шаг 4. Закончи процесс вычисления.

Легко понять, что расчленение на шаги обусловлено элементарными операциями, которые мы выбираем для решения задачи.

Применим указанный алгоритм к исходным данным $m=18$ и $n=7$.

Алгоритм начинается с шага 1, далее шаги выполняются в естественном порядке, если нет специальных указаний о порядке следования (в рассматриваемом случае такие указания есть).

Шаг 1. Установи, что $18<7$ – ложное неравенство; следуя указаниям этого шага, перейди к шагу 3.

Шаг 3. Прими m равным 18-7, т.е. 11. Перейди к шагу 1.

Шаг 1. Установи, что $11<7$ – ложное неравенство; следуя указаниям этого шага, перейди к шагу 3.

Шаг 3. Прими m равным 11-7, т.е. 4. Перейди к шагу 1.

Шаг 1. Установи, что $4<7$ – верное неравенство. Перейди к шагу 2.

Шаг 2. Прими r равным 4. Перейди к шагу 4.

Шаг 4. Закончи процесс вычисления, взяв r равным 4.

Задача 7. Буква в слове. Исполнитель – *Буквоед*. Пусть задано слово в алфавите русского языка. Относительно произвольной буквы этого алфавита требуется решить вопрос о том, входит ли она в это слово, и если входит, указать номер позиции первого вхождения этой буквы в слово. Напишите алгоритм решения задачи, учитывая, что при его выполнении Буквоед использует следующие элементарные операции: сравни две буквы, присвой значение переменной, сравни значения переменных, зафиксируй ответ, увеличь значение переменной на единицу.

Решение. Пусть $\alpha_1\alpha_2\ldots\alpha_n$ – обозначение слова, состоящего из n букв русского алфавита, а β – обозначение буквы, относительно которой требуется установить, входит она в слово или нет; i – номер позиции, занимаемой буквой α_i в слове, где i изменяется от 1 до n.

Алгоритм решения задачи может быть таким:

Шаг 1. Положи i равным 1.

Шаг 2. Сравни β с α_i. Если β совпадает с α_i, то перейди к шагу 5, иначе – к шагу 3.

Шаг 3. Увеличь *i* на единицу.

Шаг 4. Сравни *i* с *n*. Если $i \le n$, то перейти к шагу 2, в противном случае перейди к шагу 6.

Шаг 5. Зафиксируй ответ: буква β входит в *i*-ю позицию слова. Перейди к шагу 7.

Шаг 6. Зафиксируй ответ: буква β не входит в слово.

Шаг 7. Процесс закончи.

Применим исходный алгоритм к слову «алгебра» и букве «г».

Данными являются: α_1=а, α_2=л, α_3= *г*, α_4=е, α_5=б, α_6=р, α_7=а, β= *г*, *n*=7.

Каждая буква в слове занимает определенную позицию, которая указывается номером *i*, где *i* изменяется от 1 до 7.

Шаг 1. Положи *i* = 1.

Шаг 2. Сравни «г» с буквой, занимающей в слове «алгебра» первую позицию; «г» отлична от «а», перейди к шагу 3.

Шаг 3. Увеличь *i* на 1, *i* = 2. Перейди к шагу 4.

Шаг 4. Сравни *i* = 2 с 7. $2 \le 7$ – верное неравенство, перейди к шагу 2.

Шаг 2. Сравни «г» с α_2, т.е. с «л»; «г» отлична от «л», перейди к шагу 3.

Шаг 3. Увеличь *i* на 1, *i* = 3. Перейди к шагу 4.

Шаг 4. Сравни *i* = 3 с 7. $3 \le 7$ – верное неравенство, перейди к шагу 2.

Шаг 2. Сравни «г» с α_3, т.е. с «г»; «г» совпадает с α_3, перейди к шагу 5.

Шаг 5. Зафиксируй ответ. Буква *«г»* входит в третью позицию данного слова. Перейди к шагу 7.

Шаг 7. Процесс закончи.

Задача 8. Решето Эратосфена**.** Исполнитель *Вычислитель***.** Напишите алгоритм решения задачи. Из первых *n* натуральных чисел найдите все простые[3] числа. Для решения задачи примените метод, предложенный греческим математиком Эратосфеном (276-194 гг. до н.э.). Сущность метода – вычеркивается число 1, затем все числа, кратные 2 (кроме 2), затем кратные 3 (кроме 3) и т.д. Таким образом надо вы-

[3] Натуральное число, не равное 1, называется *простым*, если оно делится только на себя и на 1. Натуральное число, отличное от 1 и не являющееся простым, называется *составным.*

черкнуть все числа, кратные простым числам: $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, …, $p_k \le \sqrt{n}$.

Решение. Перечислим команды СКИ: найди неотмеченное число, отмеченным числом будем считать подчёркнутое число; определи кратность двух чисел; вычеркни (перечеркни) найденное кратное число.

Исходные данные – натуральное число *n*.

Алгоритм

Выполните последовательно действия (команды) для *n* натуральных чисел, которые меньше или равны 23.

Шаг 1. Выпиши все натуральные числа от 1 до 23, вычеркни 1.

Шаг 2. Подчеркни наименьшее натуральное число из неотмеченных, из неподчёркнутых и неперечёркнутых чисел.

Шаг 3. Вычеркни все числа, кратные подчёркнутому числу на предыдущем шаге.

Шаг 4. Проверь, имеются ли еще какие-нибудь неотмеченные числа, неподчёркнутые и неперечёркнутые, если нет, то задача решена: все подчеркнутые числа – это простые числа, иди на шаг 5. Если же имеются еще неотмеченные числа, перейди на выполнение шага 2.

Шаг 5. Процесс закончи.

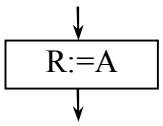
~~1~~, 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, 19, ~~20~~, ~~21~~, ~~22~~, 23

2.2. Блок-схемы[4] алгоритмов

2.2.1. Основные элементы построения блок-схем

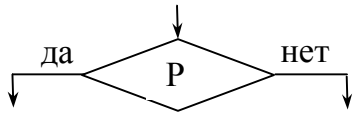
Распространенным средством записи алгоритмов является графический способ представления алгоритмов в виде блок-схем. *Блок-схема* – это ориентированный граф, в вершинах которого расположены геометрические фигуры заданного набора. Эти фигуры изображают отдельные операторы. Стрелки, соединяющие геометрические фигуры, определяют последовательность выполнения операторов.

Принято считать, что разные фигуры изображают разные операторы. Так, прямоугольник изображает преобразование информации, например, он может соответствовать оператору присваивания.

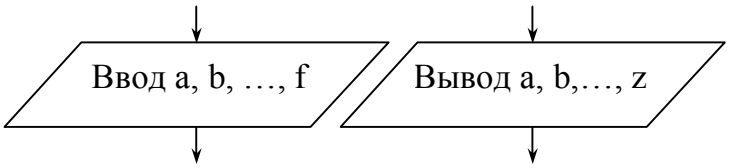


[4] Первые понятия о языке блок-схем алгоритмов ввели в 1956 году советские математики А.А. Ляпунов и Ю.Н. Янов. На Украине этими вопросами занимался в 1959 году Л.А. Калужнин.

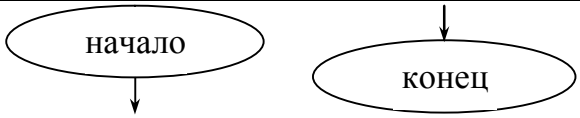
Ромб используется для изображения проверки условия



Ввод и вывод данных изображаются параллелограммом.

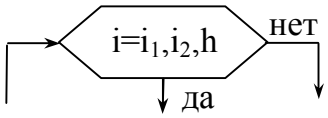


Начало и конец алгоритма изображаются овалом.

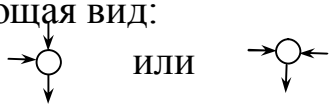


Обращение к подпрограммам, существующим в качестве самостоятельных модулей, изображается в виде прямоугольника с полями.

Для организации циклических конструкций используется блок модификации. Внутри блока записывается параметр цикла, для которого указываются его начальное и конечное значения, а также шаг изменения значения параметра для каждого повторения.



Вершина графа, имеющая вид:



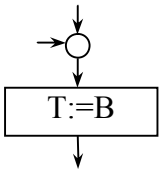
или

представляет передачу управления от одной из двух или более входящих ветвей к одной выводящей ветви. Она называется объединяющей вершиной. Объединяющую вершину используют и как указатель переноса, если блок-схема не умещается на листе.

Из блока конца алгоритма не выходит ни одной стрелки. Из остальных блоков, кроме блока проверки условия выходит только одна стрелка.

Из блока проверки условия выходит две стрелки. Стрелка с надписью «нет» (F) указывает операторы, которые выполняются, если проверяемое условие ложно (False); другая, с надписью «да» (T) указывает операторы, которые выполняются, если проверяемое условие истинно (True).

В блок «начало» не входит ни одной стрелки. В другие блоки может входить одна или несколько стрелок. Оформляется это с помощью объединяющей вершины, например,



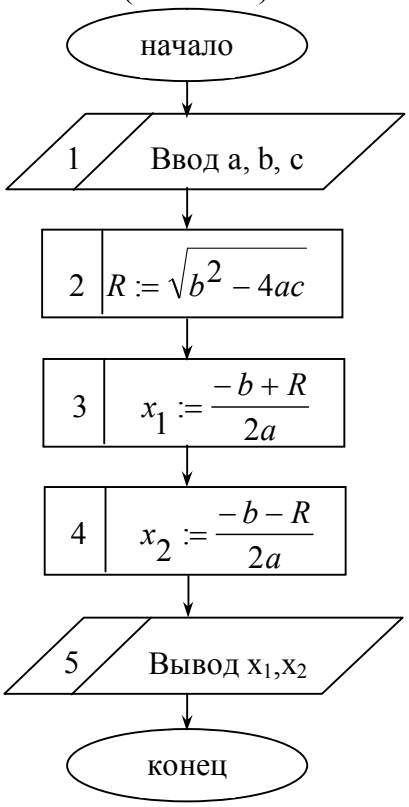
Геометрические фигуры могут быть занумерованы, т.е. иметь метки.

Задача 9. Квадратное уравнение. Составьте блок-схему алгоритма решения квадратного уравнения $a \cdot x^2 + b \cdot x + c = 0$, $a \neq 0$, если известно, что дискриминант его положителен.

Решение. Смотри блок-схему алгоритма (задача 9).

Это пример блок-схемы *линейного алгоритма*. У каждого оператора линейного алгоритма имеется только один предыдущий и один последующий оператор. Каждый из рассматриваемых операторов, кроме начала и конца – оператор преобразования информации, а потому обозначен прямоугольником.

Блок-схема алгоритма (*задача 9*):



Блок-схема алгоритма имеет следующие *достоинства*:

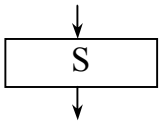
1. Она способствует лучшему зрительному восприятию структуры алгоритма, облегчает его запоминание.

2. Видна логическая последовательность действий; следуя в направлениях, указанных стрелками, наглядно представляется работа всей блок-схемы.

2.2.2. Основные управляющие команды организации действий в алгоритмах

Доказано, что организовать действия в любой задаче можно, используя комбинации основных управляющих структур: функцио-

нального блока, условных структур, циклических структур. В приведенных ниже блок-схемах основных управляющих структур организации действий в алгоритмах приняты такие обозначения: P, P_1, … – проверяемые условия (логические выражения); S, S_1, S_2, … – действия по обработке информации.

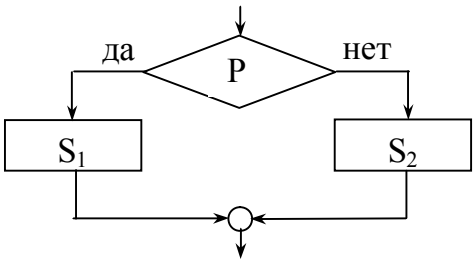


1. *Функциональный блок* имеет вид:

S может быть оператором, командой вызова с возвратом некоторой процедуры, другой управляющей структурой организации действий в алгоритме.

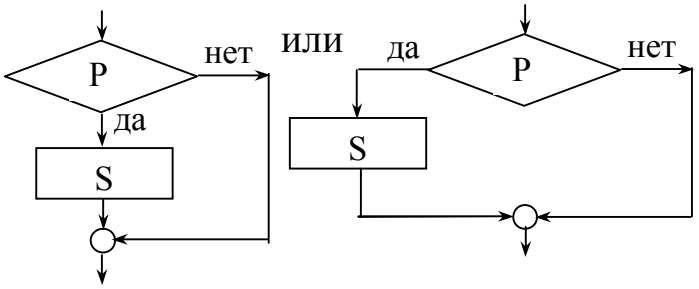
2. *Условные структуры*.

а) Команда ветвления в полной форме – “if P then S_1 else S_2”, “если P то S_1 иначе S_2” имеет вид:



В этой команде сначала проверяется истинность логического выражения P. Если P истинно, то выполняются операторы, определяемые функциональным блоком S_1 и далее оператор, следующий за командой ветвления, а S_2 игнорируется; если P ложно, то выполняются операторы, определяемые функциональным блоком S_2 и далее оператор, следующий за командой ветвления, а S_1 игнорируется.

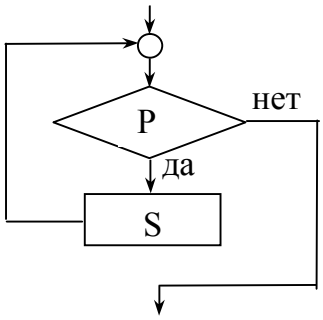
б) Команда ветвления в сокращенной (неполной) форме –“if P then S”, “если P то S” имеет вид:



Эту команду можно рассматривать как сокращенную запись структуры 2а), если отсутствует функциональный блок S_2. Если выражение P истинно, то выполняются операторы, определяемые функциональным блоком S и далее операторы, следующие за командой ветвления, если ложно, то выполняется оператор, непосредственно следующий за командой ветвления.

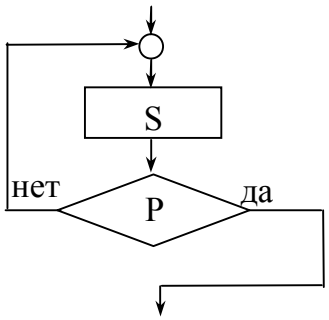
3. *Циклические структуры*.

а) Команда повторения «пока» – “while P do S”, “пока P истинно выполняй S”, “цикл с предусловием” имеет вид:



По этой команде сначала проверяется условие P. Если оно выполняется, то производятся действия, определяемые функциональным блоком S, затем снова проверяется условие P и так далее. При невыполнении условия P (по стрелке с надписью «нет») происходит выход из цикла, далее выполняются операторы, следующие за командой повторения.

б) Команда повторения “repeat S until P”, “повторять S до P”, “цикл с постусловием” (повторять S, пока P не станет истинным) имеет вид:

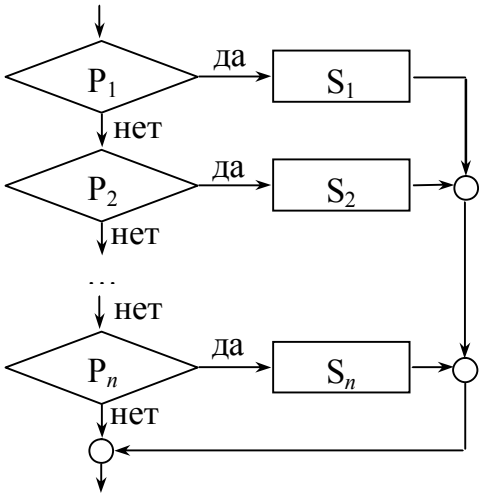


Эта организация цикла такова, что сначала выполняются операторы, определяемые функциональным блоком S, а затем проверяется необходимость его повторения.

2.2.3. Дополнительные управляющие команды организации действий в алгоритмах

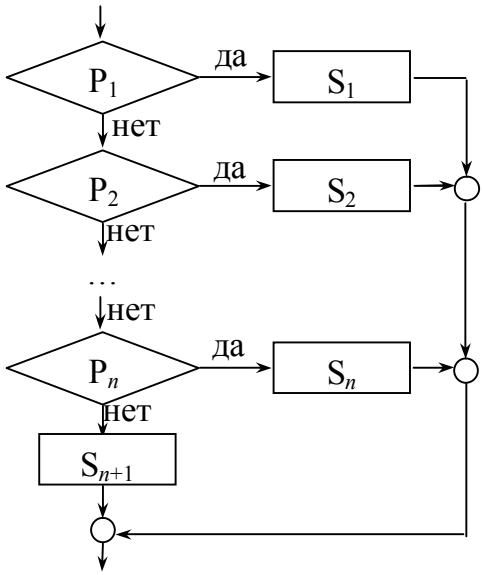
Если при решении задачи необходимо выбрать один из многих вариантов действий, то для написания алгоритма решения такой задачи используется управляющая команда выбора в полной или неполной форме.

1. *Выбор в неполной форме*.



По этой команде сначала проверяется условие P_1. Если оно выполняется, то производятся действия, определяемые функциональным блоком S_1 и далее оператор, следующий за командой выбора; если P_1 ложно, то проверяется условие P_2. Если условие P_2

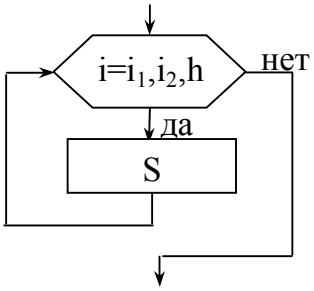
истинно, то производятся действия, определяемые функциональным блоком S_2 и далее оператор, следующий за командой выбора; если P_2 ложно, то проверяется условие P_3 и т.д. до P_n. Если P_n выполняется, то производятся действия, определяемые функциональным блоком S_n и далее оператор, следующий за командой выбора; если P_n ложно, то команда выбора никаких действий не предписывает и сразу выполняется следующий за ней оператор.



2. *Выбор в полной форме*.

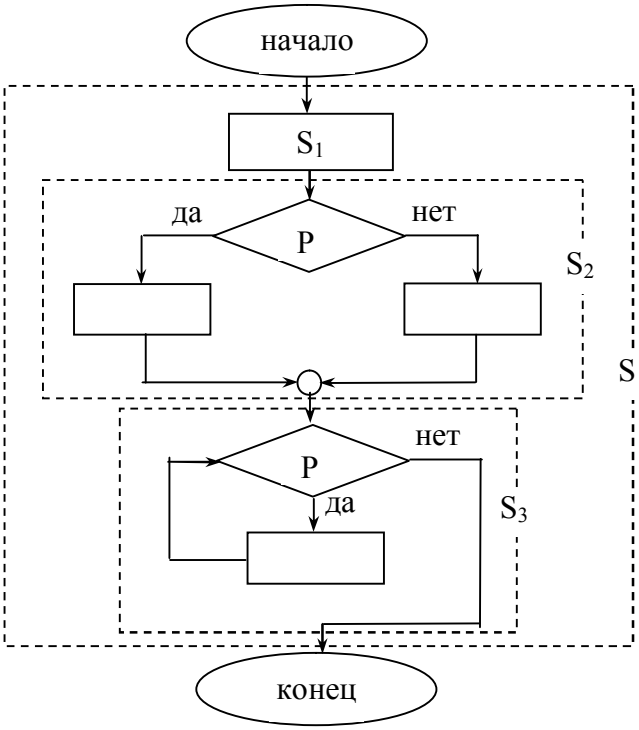
Так же как и в выборе в неполной форме, по этой команде сначала проверяется условие P_1. Если оно выполняется, то производятся действия, определяемые функциональным блоком S_1 и далее оператор, следующий за командой выбора в полной форме; если P_1 ложно, то проверяется условие P_2 и т.д. Если условие P_n выполняется, то производятся действия, определяемые функциональным блоком S_n, и далее выполняется оператор, следующий за командой выбора в полной форме; если условие P_n ложно, то производятся действия, предписываемые функциональным блоком S_{n+1}, и далее выполняется оператор, следующий за командой выбора в полной форме.

3. *Цикл «для», цикл с параметром*.

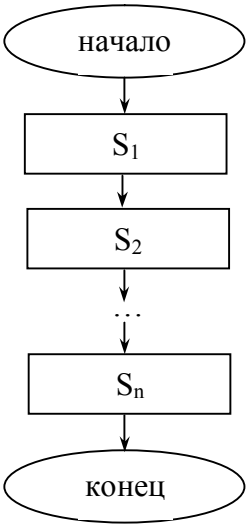


Команда предписывает выполнять действия, которые содержит функциональный блок S, для всех значений некоторой переменной *i* (параметра цикла) в заданном диапазоне от i_1 до i_2 с шагом *h*.

Перечисленные структуры (команды) имеют один вход и один выход (одна входящая и одна выходящая стрелка). *Структуры* и *любые комбинации структур*, имеющие один вход и один выход, могут *рассматриваться как один функциональный блок*.



Решение любой задачи можно представить в виде цепочки (линейной комбинации базовых структур):



С другой стороны, отдельные функциональные блоки могут быть заменены одной из базовых структур или их комбинацией.

Запись алгоритмов с использованием основных структурных элементов улучшает читаемость блок-схем и программ, облегчает общение между программистами, делает возможным доказательство некоторых свойств программ.

3. Примеры блок-схем алгоритмов

3.1. Блок-схемы алгоритмов, содержащих команды ветвления

Задача 10. Большее из двух чисел. Составьте блок-схему алгоритма нахождения большего из двух чисел a и b; переменой x присвойте значение $\max(a, b)$.

Решение. Смотри блок-схему алгоритма (задача 10).

В зависимости от знака R, т.е. от истинности логического выражения $R<0$, решение может идти по одному из двух возможных направлений (разветвляться). Если $R<0$ – ложь, то после оператора 3 выполняется оператор 6, к которому направлена стрелка с надписью «нет», в этом случае при решении задачи будут выполнены операторы 1, 2, 3, 6, 5, 7. Если $R<0$ – истина, то после оператора 3

будет выполнен оператор 4, к которому направлена стрелка с надписью «да», а затем операторы 5 и 7.

Задача 11. Квадратное уравнение. Составьте блок-схему алгоритма нахождения корней квадратного уравнения $a \cdot x^2 + b \cdot x + c = 0$, $a \neq 0$, если они действительные и различные, иначе корни не вычислять.

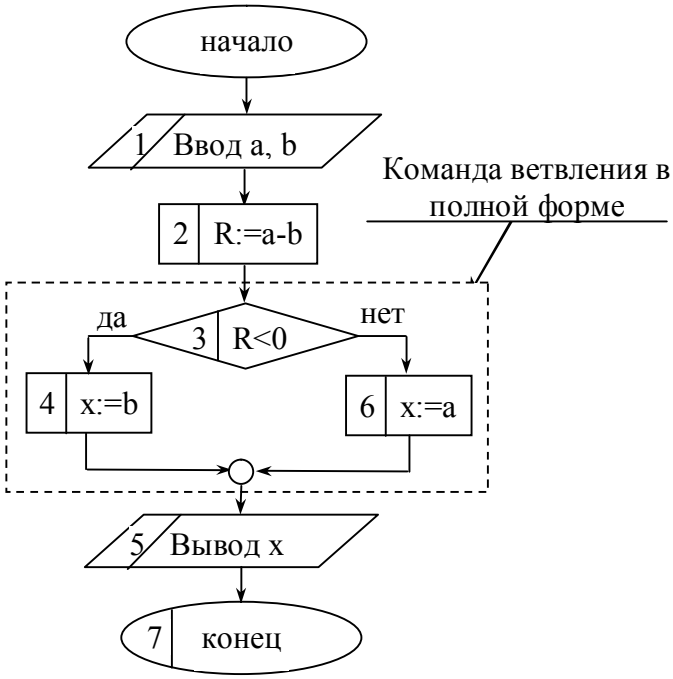
Решение. Смотри блок-схему алгоритма (задача 11).

В этой задаче корни квадратного уравнения x_1 и x_2 вычисляются, если $D>0$, тогда выполняются операторы 1-8. Если $D \leq 0$, то выполняются операторы 1-3 и 8.

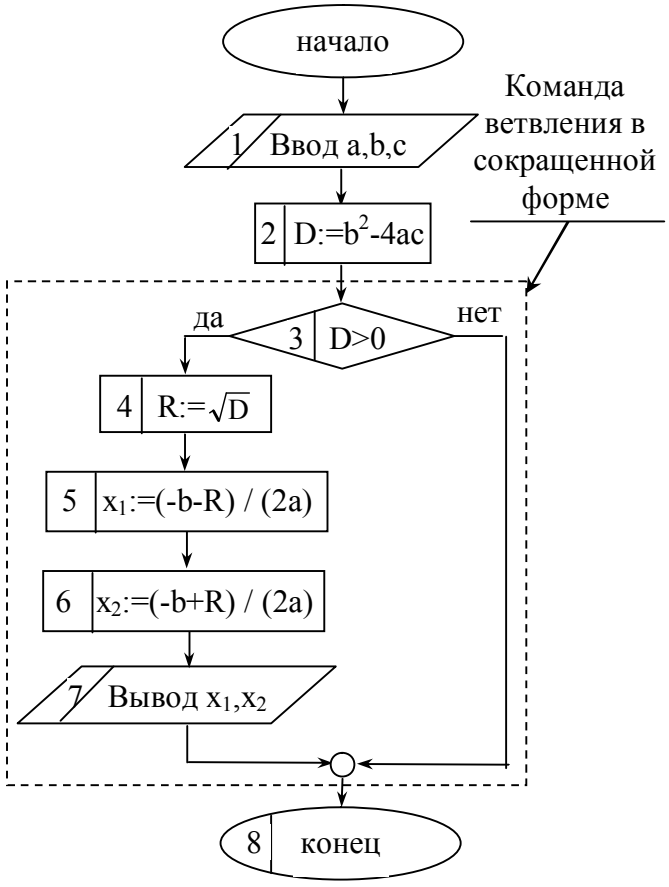
Задача 12. Площадь треугольника. Даны длины трех отрезков *a*, *b*, *c*. Если существует треугольник, сторонами которого являются данные отрезки, то вычислите его площадь *S*. Иначе переменной *S* присвойте значение «-1». Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 12).

Блок-схема алгоритма (*задача 10*):



Блок-схема алгоритма (*задача 11*):

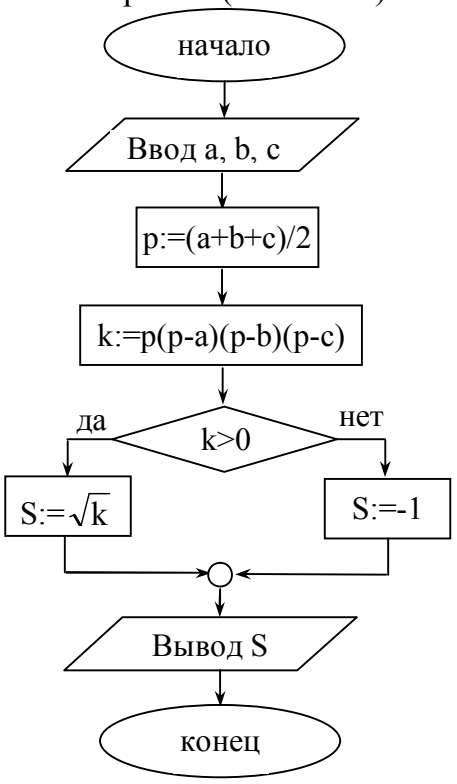


Можно показать, что если $p\cdot(p-a)\cdot(p-b)\cdot(p-c)>0$, то треугольник со сторонами a, b, c существует.

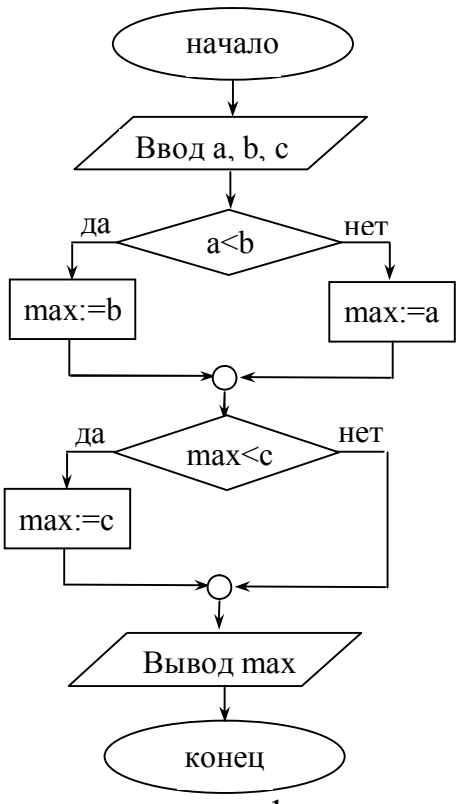
Задача 13. Большее из трех чисел. Составьте блок-схему алгоритма нахождения большего из чисел a, b, c. Переменной *max* присвойте значение большего из трех чисел.

Решение. Смотри блок-схему алгоритма (задача 13).

Блок-схема алгоритма (*задача 12*):



Блок-схема алгоритма (*задача 13*):



Задача 14. Значение функции. Вычислите значение функции

$$y(x)=\begin{cases} x\text{-}1, \text{ если } x\le 3,\\ x^3+2, \text{ если } 3<x\le 22,\\ \dfrac{1}{x^2+1}, \text{ если } x>22. \end{cases}$$

Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схемы 1-3 алгоритма (задача 14).

Если задать функцию в виде:

$$y(x)=\begin{cases} F(x), \text{ если } x\le 22,\\ \dfrac{1}{x^2+1}, \text{ если } x>22, \end{cases}$$

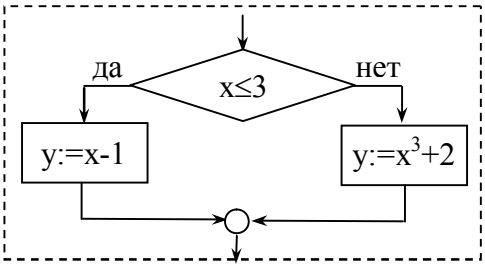
где

$$F(x)=\begin{cases} x\text{-}1, \text{ при } x \le 3, \\ x^3+2, \text{ при } 3 < x \le 22, \end{cases}$$

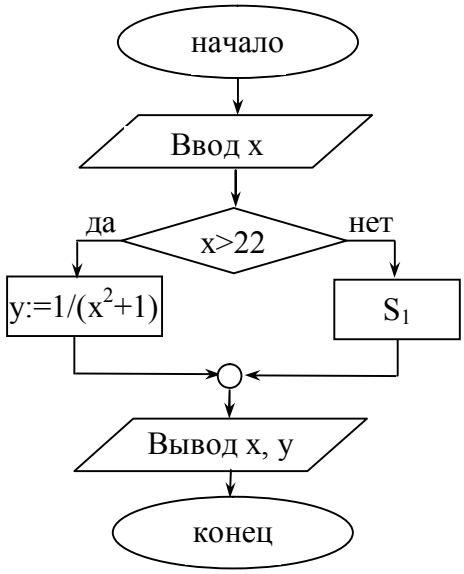
то составление блок-схемы можно провести в два этапа.

1 этап. Вычисление значения функции $F(x)$ опишем функциональным блоком S_1, тогда блок-схема вычисления значения функции $y(x)$ – блок-схема 1 алгоритма (задача 14)

В свою очередь S_1 – команда ветвления в полной форме:



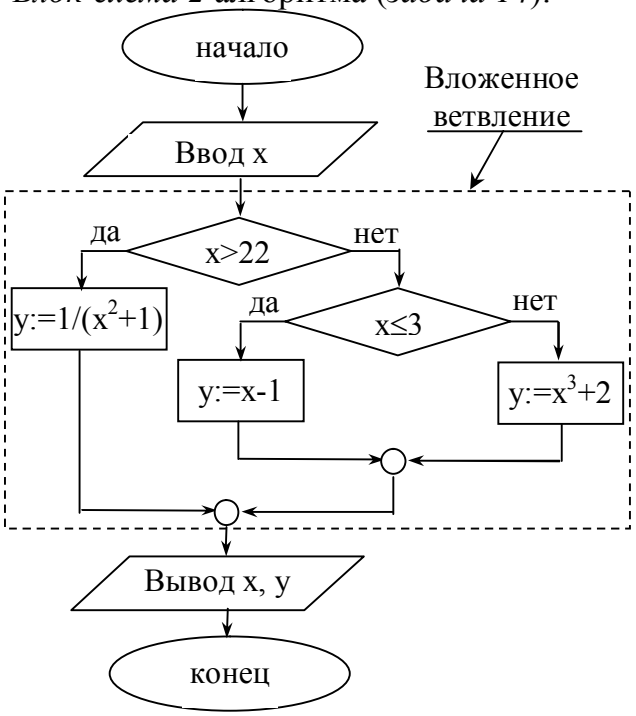
Блок-схема 1 алгоритма (*задача 14*):



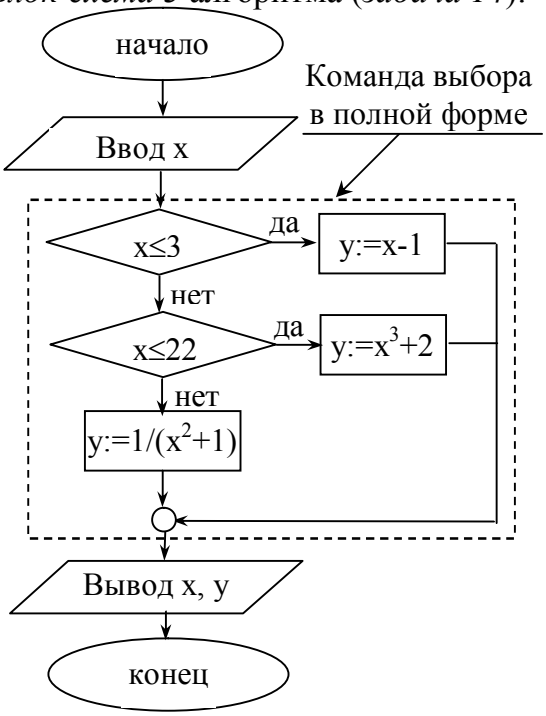
2 этап. Подставив детализированный блок S_1 в блок-схему 1 (задача 14), получим блок-схему 2 алгоритма (задача 14) или блок-схему 3 алгоритма (задача 14).

В блок-схеме 2 алгоритма (задача 14) для записи алгоритма использовано вложенное ветвление, а в блок-схеме 3 алгоритма (задача 14) – команда выбора в полной форме.

Блок-схема 2 алгоритма (*задача 14*):



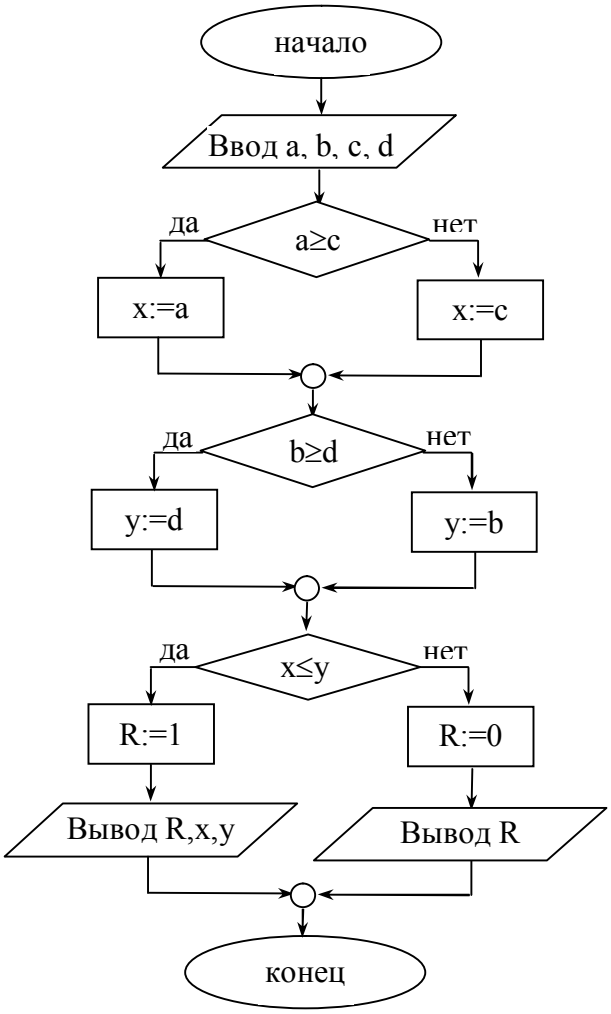
Блок-схема 3 алгоритма (*задача 14*):



Задача 15. Пересечение отрезков. На координатной прямой даны два отрезка [*a*;*b*] и [*c*;*d*]. Если отрезки пересекаются, то укажите концы отрезка, являющегося их пересечением и переменной *R* присвойте значение 1, иначе *R* присвойте значение 0. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 15).

Блок-схема алгоритма (*задача 15*):



3.2. Блок-схемы алгоритмов, содержащих команды повторения

Алгоритмы, описывающие процессы, в которых одни и те же действия выполняются многократно в одном месте алгоритма при различных значениях переменных, называются циклическими. Последовательность многократно исполняющихся операторов называют телом цикла. Циклы являются составными частями многих, практически реализуемых алгоритмов. Использование циклов позволяет существенно сократить схему алгоритма и длину соответствующей ему программы. При записи циклических алгоритмов используют команды повторения, стр. 33, 34.

Задача 16. Сумма кубов. Составьте блок-схему алгоритма вычисления суммы кубов натуральных чисел от 1 до 100, $S = \sum_{n=1}^{100} n^3$.

Решение. Смотри блок-схему алгоритма (задача 16).

Сумму S можно находить последовательно так: $S:=1^3$; $S:=S+2^3$; $S:=S+3^3$ и т.д.

Вместо подобной цепочки операторов принято писать: $S:=S+n^3$, где $1 \le n \le 100$.

Чтобы при первом выполнении этого оператора значение суммы было равно 1, нужно положить начальное значение S равным 0.

Натуральные числа и их кубы получаются в результате работы операторов присваивания, поэтому в данном случае оператор ввода отсутствует. При решении задачи многократно повторяются операторы 3, 4, 5. Операторы 3, 4 составляют тело цикла данного алгоритма. Поскольку значение целой переменной n связано с числом повторений, её называют параметром или счетчиком цикла.

Первый раз цикл выполняется при значении n равном 1; это значение называется начальным значением параметра цикла. При повторении цикла значение n изменяется на шаг h, в данном случае $h = 1$. Конечное значение параметра цикла равно 100.

Блок-схема алгоритма (*задача 16*):

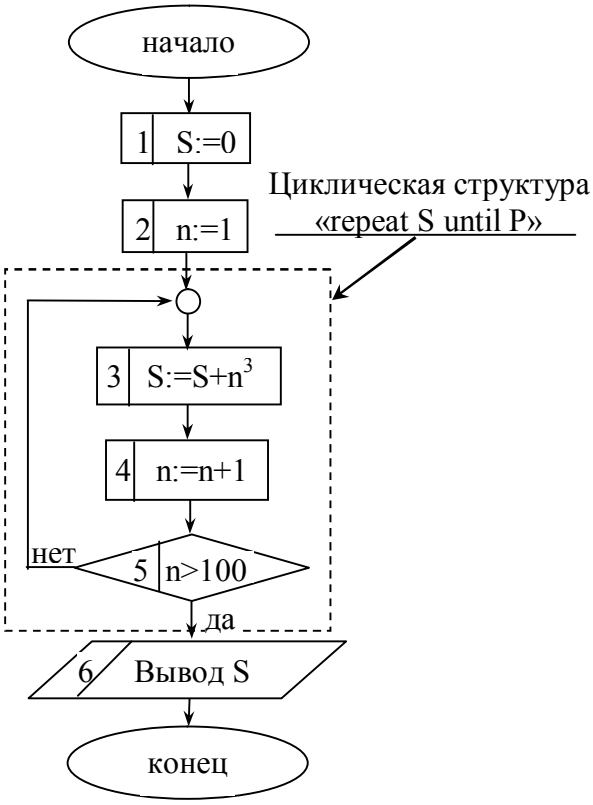
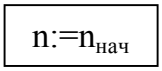
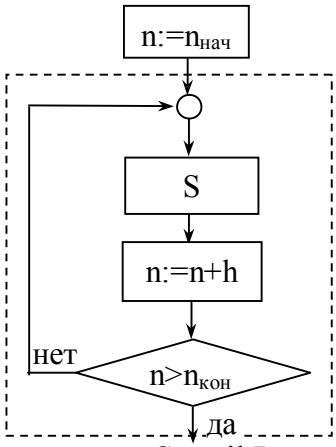


Схема структуры, содержащей цикл, управляемый счетчиком

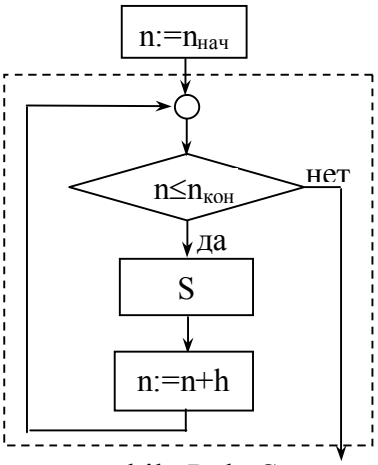
Перед циклической структурой в блок-схеме алгоритма, содержащего цикл, управляемый счетчиком, ставиться оператор присваивания параметру цикла начального значения:



Циклические структуры в этом случае можно детализировать так:



«repeat S until P»



«while P do S»

Задача 17. Корень третьей степени. Составьте блок-схему алгоритма вычисления приближенного значения $y = \sqrt[3]{x}$ с точностью до ε по заданной итерационной формуле: $y_{n+1} = \frac{2}{3}\left(y_n + \frac{x}{2 \cdot y_n^2}\right)$.

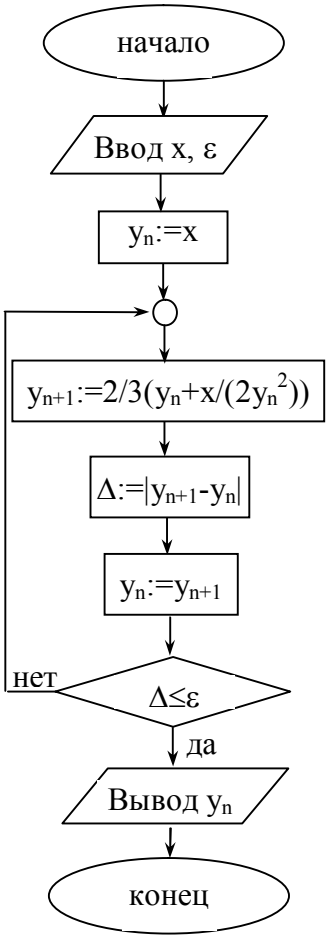
Процесс считать оконченным, если $|y_{n+1}-y_n| \le \varepsilon$, где y_n – предыдущее (в том числе, и начальное), а y_{n+1} – последующее приближения.

Решение. Смотри блок-схему алгоритма (задача 17).

Обозначим $\Delta = |y_{n+1}-y_n|$. За начальное приближение y_n возьмем, например, x. Проверка окончания определяется значением выражения $\Delta \le \varepsilon$. Если $\Delta \le \varepsilon$ – истинно, то вычисления заканчиваем, если ложно, находим следующее приближение, приняв за исходное значение y_n значение y_{n+1}. В блок-схеме это реализуется оператором присваивания $y_n := y_{n+1}$

Это пример итерационного цикла[5]. В итерационных циклах

Блок-схема алгоритма (*задача 17*):



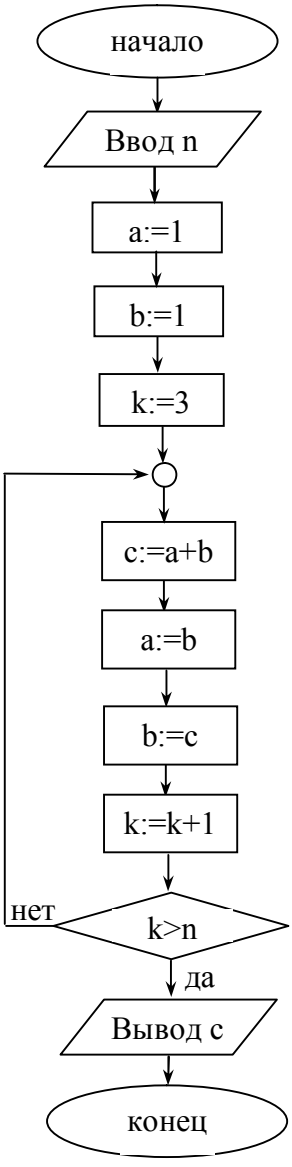
[5] Итерация – повторение.

результат получают методом последовательных приближений, который состоит в повторении группы операций таким образом, что исходными данными для каждого следующего вычисления являются результаты предыдущего вычисления.

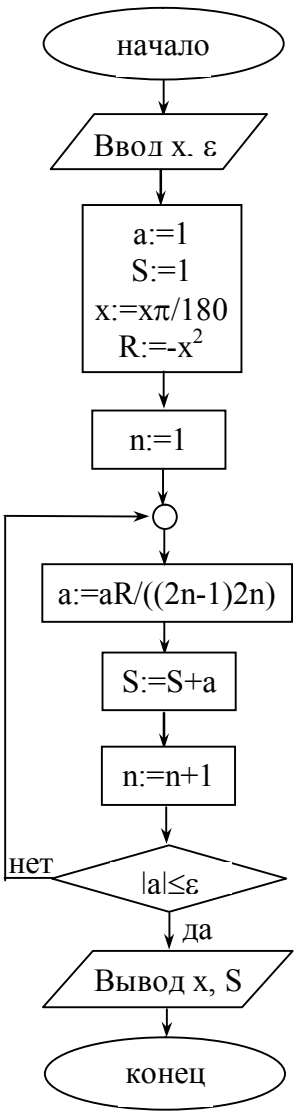
Задача 18. Число Фибоначчи[6]. Члены последовательности Фибоначчи определяются соотношениями: $a_1 = 1$, $a_2 = 1$, $a_k = a_{k-1}+a_{k-2}$, $k \geq 3$. Найдите n-ый член этой последовательности, где $n \geq 3$. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 18).

Блок-схема алгоритма (*задача 18*):



Блок-схема алгоритма (*задача 19*):



[6] Итальянский математик Леонардо Пизанский (Фибоначчи) (1170-1250).

Поскольку промежуточные значения элементов последовательности нас не интересуют, индексы элементов можно опустить. В связи с этим введены обозначения: *c* – *k*-ый элемент, *b* – (*k*-1)-ый, *a* – (*k*-2)-ой элемент последовательности.

Задача 19. Косинус угла. Составьте блок-схему вычисления значения функции $y = \cos x$ с точностью до ε для заданного в градусах x.

Решение. Смотри блок-схему алгоритма (задача 19).

Для вычисления значения cos *x* применим разложение его в ряд Тейлора:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2 \cdot n)!} + \ldots$$

Если обозначить предыдущий член ряда a_n, последующий a_{n+1}, то

$$a_{n+1} = a_n \frac{-x^2}{(2 \cdot n-1) \cdot 2 \cdot n}.$$

Вычисления по этой формуле будут многократно повторяться при новых значениях *n*, цикл будет итерационным.

Известно, что для монотонных знакопеременных рядов вычисление значения функции с данной точностью можно закончить, когда будет найден член ряда, по модулю не превосходящий заданную точность.

Поскольку нет необходимости запоминать значения предыдущего и последующего членов ряда, формула вычисления a_{n+1} может быть упрощена:

$$a := a \frac{-x^2}{(2 \cdot n-1) \cdot 2 \cdot n}.$$

Начальное значение *a*, очевидно, равно 1; начальное значение переменной *S*, которая будет принимать значение косинуса угла, также равно 1.

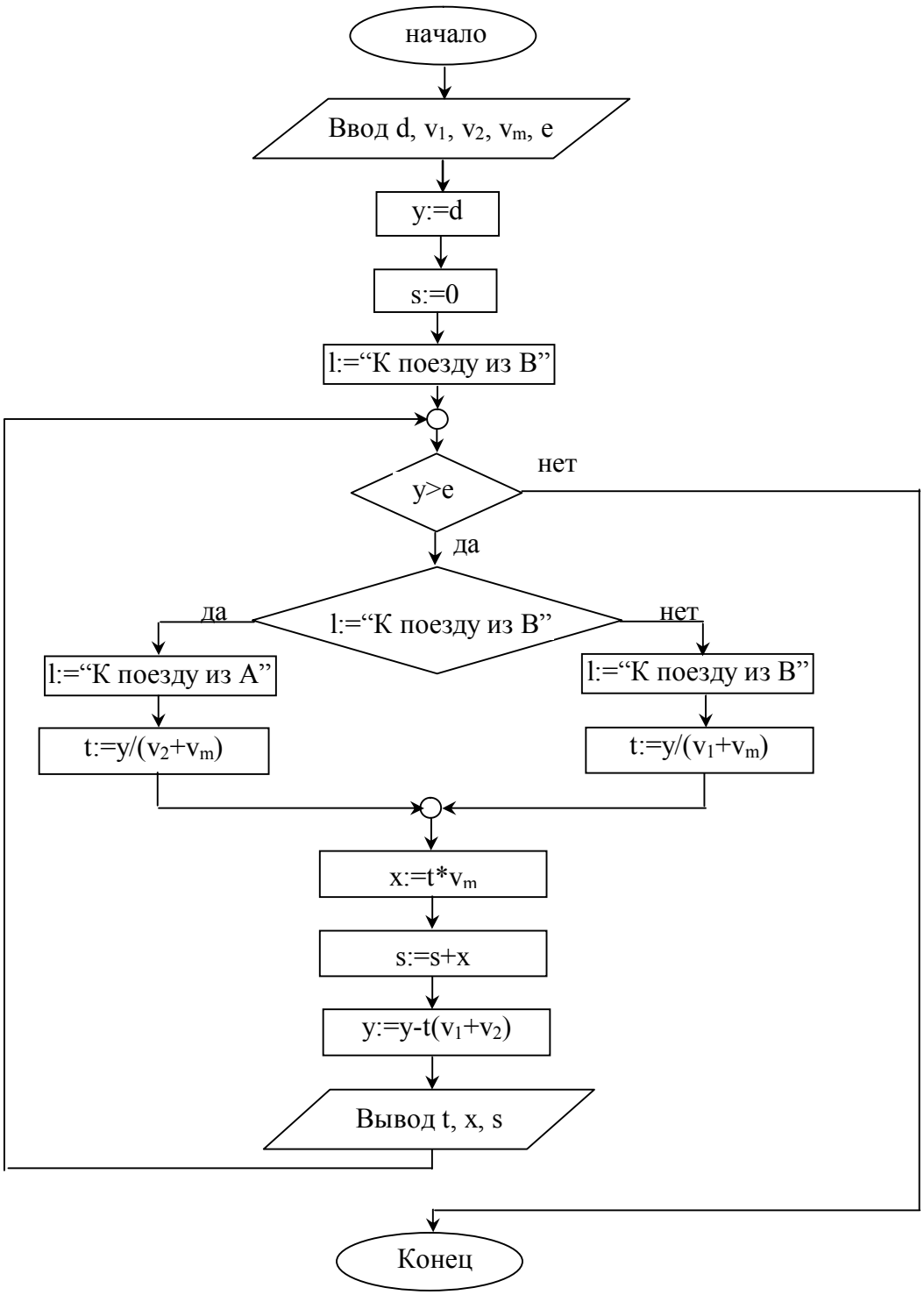
Поскольку аргумент задан в градусах, необходимо перевести его в радианы по формуле: $x = \frac{x \cdot \pi}{180}$.

Задача 20. Сверхбыстрая муха. Из пункта A и пункта B, удаленных друг от друга на расстояние *d* км, навстречу друг другу одновременно отправляются два поезда, скорость первого – v_1 км/ч, скорость второго – v_2 км/ч. В то же время из пункта A вылетает сверхбыстрая муха со скоростью v_m км/ч и летит навстречу поезду, вышедшему из пункта B; встретившись с ним,

она летит к поезду, вышедшему из пункта A и т.д. до тех пор, пока поезда не встретятся. Опишите алгоритм нахождения: а) длины отрезков пути, которые пролетает муха до встречи с очередным поездом; б) общее расстояние, которое пролетит муха, с точностью до *e*.

Решение. Смотри блок-схему алгоритма (задача 20).

Блок-схема алгоритма (*задача 20*):



На второй вопрос задачи можно ответить так: поезда встретятся через t часов, где $t = d/(v_1+v_2)$, а муха пролетит за это же время расстояние $s = v_m \cdot t$ (км). Пусть d = 600 км, v_1 = 40 км/ч, v_2 = 60 км/ч, v_m = 200 км/ч, тогда муха пролетит расстояние s = 200·600/(40+60) = 1200 (км). Тем не менее, будем отвечать на второй вопрос задачи, используя результаты решения первой части задачи.

Пусть y – расстояние, которое в момент встречи с одним поездом, отделяет муху от другого поезда в самом начале нового отрезка пути. Тогда:

- если встречным поездом будет поезд из В, то продолжительность полета равна $t = y/(v_m+v_2)$, следовательно, мухе надо пролететь расстояние $x = t \cdot v_m$;
- если встречным поездом будет поезд из А, то вычисления проводятся по следующим формулам $t = y/(v_m+v_1)$, $x = t \cdot v_m$.

Вначале расстояние равно d ($y=d$). Когда муха встретит поезд после полета в течение времени t, ее будет отделять от другого поезда расстояние $y = y - t(v_1+v_2)$.

3.3 Блок-схемы алгоритмов работы с массивами

Используя команды повторения можно коротким алгоритмом описать большой объем действий, необходимых для решения поставленной задачи. Массивы играют аналогичную роль по отношению к данным и позволяют коротким алгоритмом описать действия с огромным объемом информации.

Применение массивов при описании алгоритмов решения практических задач позволяет:

1. записать коротким алгоритмом работу с большим объемом информации;
2. расширить область применимости алгоритма.

Необходимость в применении массивов для описания алгоритмов решения задач возникает, когда при решении задач приходится иметь дело с большим, конечным количеством однотипных упорядоченных по индексам объектов.

Массив – это упорядоченная по индексам, конечная совокупность однотипных объектов, образованных по одному и тому же правилу. Отношение порядка в массиве задается с помощью индексирования элементов массива. Если для индексирования массива используется один индекс, то массив называется одномерным, если два или больше, то многомерным. Для индексации элементов двумерного массива указываются два индекса, первый индекс, как правило, номер

строки, второй – номер столбца. Одномерный массив часто называют вектором, двумерный – матрицей.

При работе с массивами, каждому массиву дается имя. Работа с массивом – это работа с элементами массива. Элементу массива дается имя, соответствующее имени массива, и указывается в квадратных или круглых скобках порядковый номер этого элемента в массиве.

Очевидно, чтобы задать массив (таблицу), необходимо:

1. указать, что однотипные объекты объединены в массив (таблицу);

2. указать имя массива (таблицы), начальный и конечный порядковые номера индексов его (её) элементов;

3. указать тип значений элементов массива (таблицы).

При описании массива после имени массива будем в круглых (квадратных) скобках указывать начальный и конечный номера каждого индекса элементов массива через двоеточие. Если массив многомерный, то описание начального и конечного номеров каждого индекса элементов массива разделим запятой. Например, A(1:50) – массив, элементы которого: A(1), A(2), … , A(50); B(1:2,1:3) – массив, элементы которого: $\begin{pmatrix} B(1,1) B(1,2) B(1,3) \\ B(2,1) B(2,2) B(2,3) \end{pmatrix}$.

Пример 1. Одномерный массив Осадки (1:365) – количество осадков в течение года.

Дни года	1	2	3	…	364	365
Осадки в мм	50	34	120	…	23	5

Пример 2. Двумерный массив Расписание (1:4,1:2) – расписание уроков на 2 дня в 4 классе общеобразовательной школы.

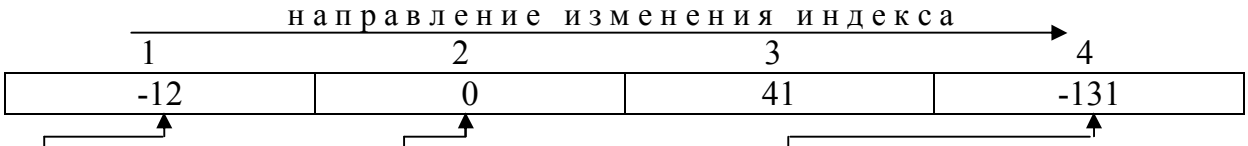
Дни недели / Номер урока	Понедельник	Вторник
1	Математика	Русский язык
2	Русский язык	Математика
3	Природоведение	История
4	Физкультура	Рисование

Массивы имеют размер и размерность. *Размер массива* – это *количество элементов* в данном массиве, *размерность – количество*

индексов, необходимых для однозначного определения места фиксированного элемента массива. Массив примера 1 имеет размерность, равную 1 (одномерный), размер – 365. Массив примера 2 имеет размерность равную 2 (двумерный), размер 2·4=8. Элемент массива называется переменной с индексом, переменная без индекса – простой переменной. Над элементами массива можно производить те операции, которые допустимы для базового типа.

В качестве типов индексов элементов массива можно использовать целый тип. Индексы могут задаваться константами, переменными и выражениями. Значения переменных и выражений задают номер элемента массива, поэтому их значения должны быть определены при обращении к этому элементу. Элементами массива могут быть значения любого типа данной реализации языка.

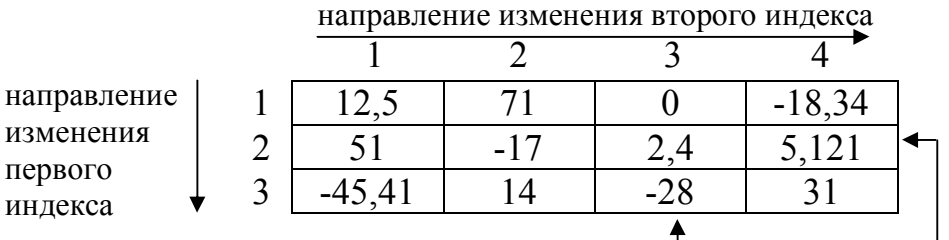
Пример 3. Пусть массив *A* – одномерный массив, имеющий 4 элемента целого типа – integer: -12, 0, 41, -131.



A[1]=-12; если *i*=2, то A[*i*]=0; если *i*=1, *j*=3, то A[*i*+*j*]=-131

Пример 4. Массив *Q* – двумерный массив, имеющий 3 строки и 4 столбца – 12 элементов вещественного типа – real:

$$\begin{pmatrix} 12{,}5 & 71 & 0 & -18{,}34 \\ 51 & -17 & 2{,}4 & 5{,}121 \\ -45{,}41 & 14 & -28 & 31 \end{pmatrix}.$$



Q[3,3]=-28; если *i*=1, *j*=2, *s*=2, то Q[*i*·*s*,*j*+2]=Q[2,4]=5,121

Задача 21. Произведение элементов массива. Составьте блок-схему алгоритма нахождения произведения вещественных чисел

$$П = a(1)\cdot a(2)\cdot\ldots\cdot a(n).$$

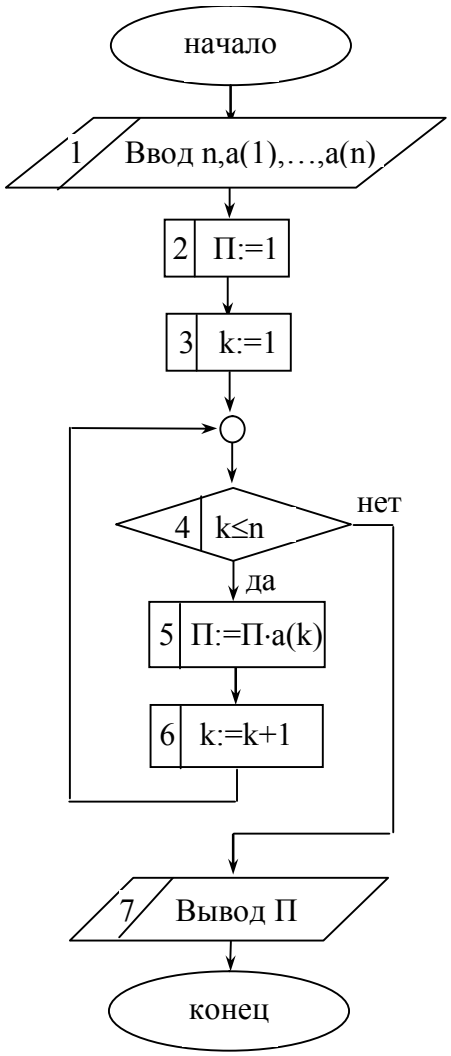
Решение. Смотри блок-схему алгоритма (задача 21), использована циклическая структура «while P do S».

Произведение можно находить так:

П:=a(1); П:=П·a(2); … ; П:=П·a(n).

В блок-схеме вместо этой цепочки операторов запишем оператор П:=П·a(k), где k изменяется от 1 до n; k – параметр цикла. Чтобы значение первого произведения было равно a_1, следует положить начальное значение П равным 1. Операторы 5-6 составляют тело цикла.

Блок-схема алгоритма (*задача 21*):



Задача 22. Произведение матриц. Составьте блок-схему нахождения произведения матрицы A на матрицу B:

$$A=\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, \qquad B=\begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}.$$

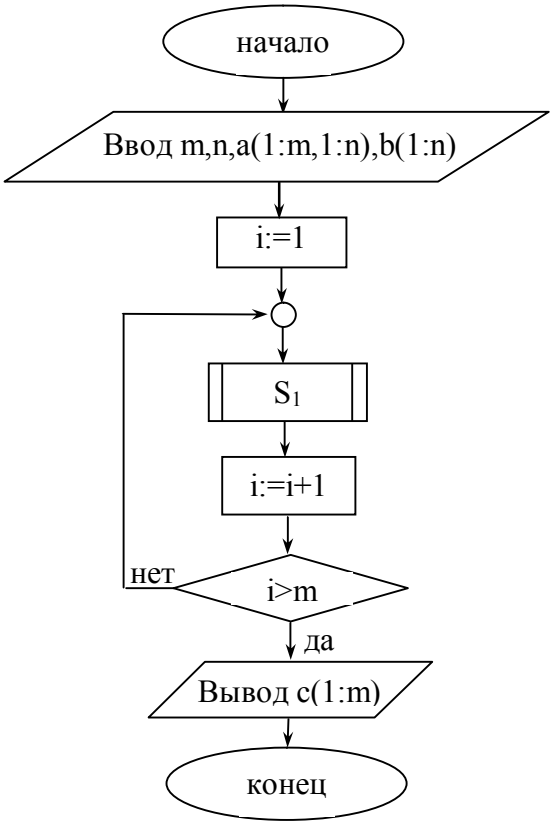
Решение. Смотри блок-схемы 1-3 алгоритма (задача 22).

Смотри блок-схему 1 (задача 22). Произведение матрицы A на матрицу B есть матрица-столбец; обозначим ее C.

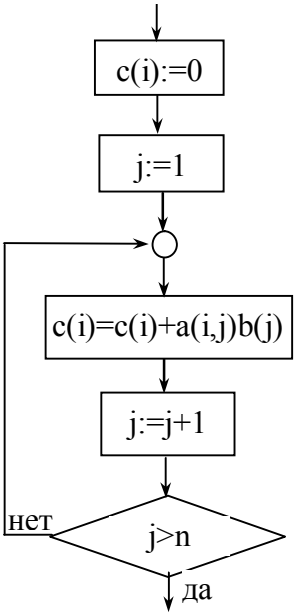
$$C=\begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_m \end{pmatrix}, \text{ где } c(i)=a(i,1)\cdot b(1)+a(i,2)\cdot b(2)+\ldots+a(i,n)b(n)=\sum_{j=1}^{n} a(i,j)\cdot b(j),\ i=1,2,\ldots,m.$$

Функциональный блок, вычисляющий величину c(i), обозначим S_1. Смотри блок-схему 2 (задача 22).

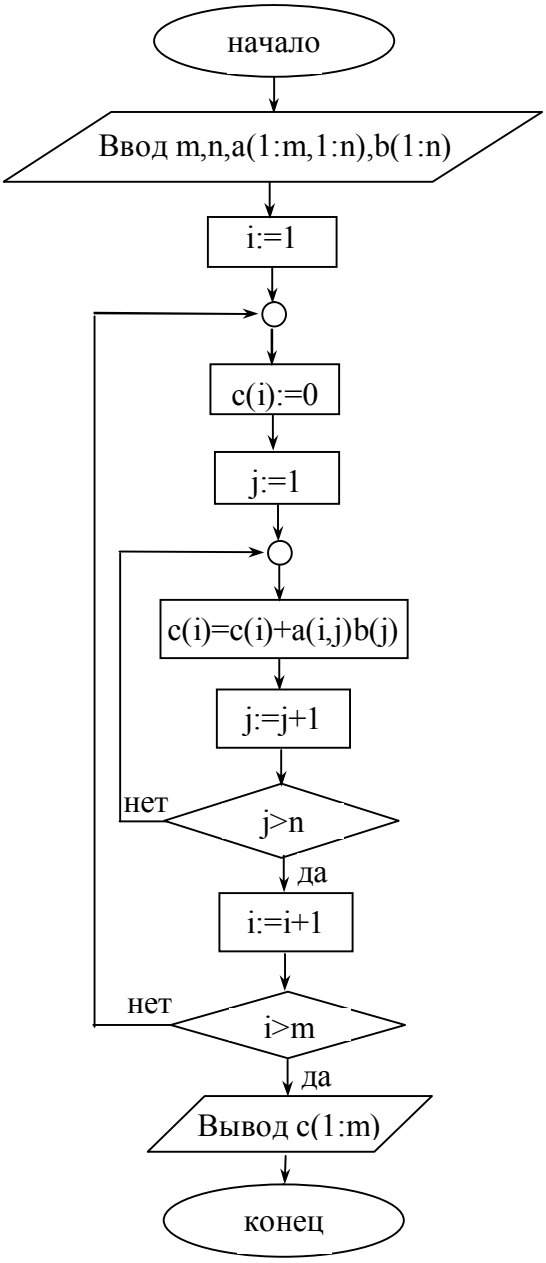
Блок-схема1 алгоритма (*задача 22*):



Блок-схема2 алгоритма (*задача 22*):



Блок-схема3 алгоритма (*задача 22*):



Получать элементы c(i) при фиксированном i можно так: c(i) = c(i)+a(i,j)·b(j), где $1 \le j \le n$, а начальное значение c(i) равно 0.

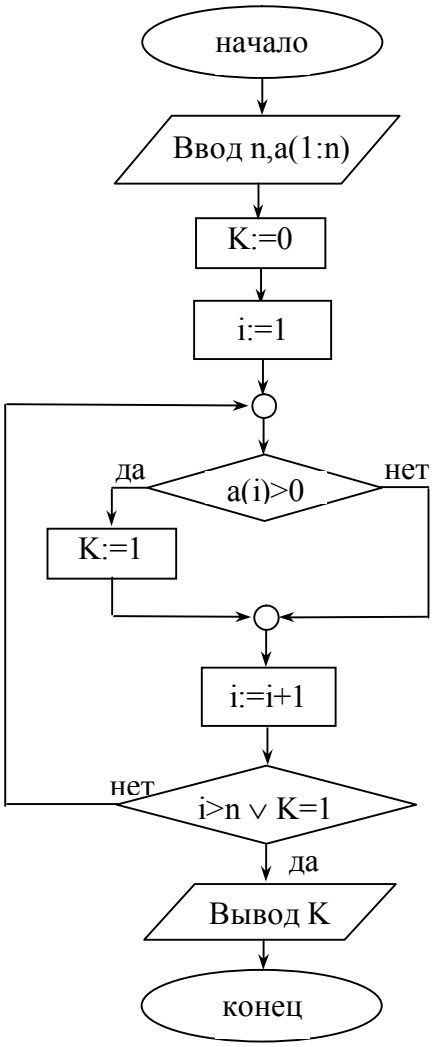
Очевидно, что функциональный блок S_1 будет содержать циклическую структуру, например, структуру «repeat S until P».

Подставив детализацию блока S_1 (блок-схема 2 (задача 22)) в блок-схему 1 (задача 22), получим одну циклическую структуру внутри другой, вложенные циклы – блок-схема 3(задача 22).

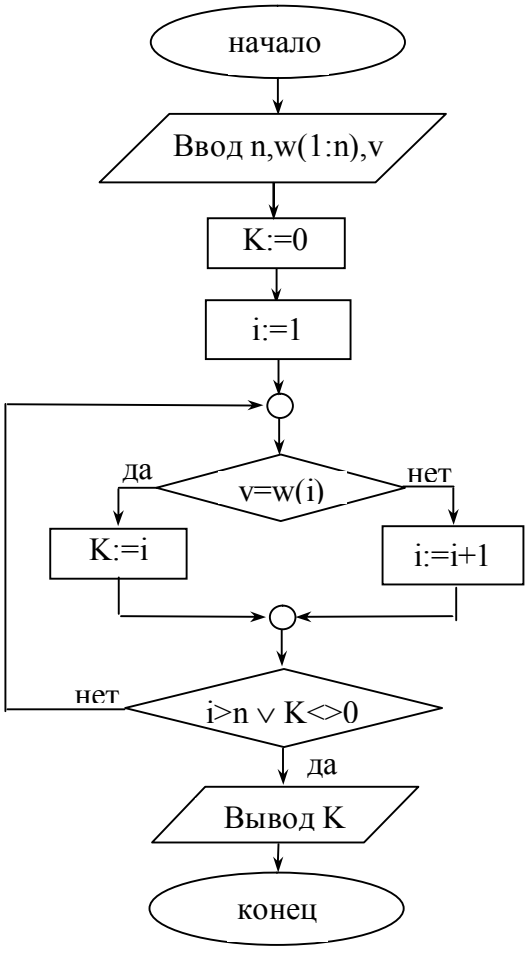
Задача 23. Положительные числа. Дана числовая последовательность a_1, a_2, … , a_n. Определите, есть ли среди a_i, $1 \le i \le n$ положительные числа, если да, то переменной K присвойте значение 1, в противном случае – значение 0. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 23).

Блок-схема алгоритма (*задача 23*):



Блок-схема алгоритма (*задача 24*):



Поскольку положительный элемент может оказаться на i-ом месте, где $i<n$, то нужно иметь возможность выйти из цикла при $i<n$. С этой целью перед началом цикла переменной K присвоено значение 0, а если $a_i>0$ – истинно, K получит значение 1. Проверка окончания организована так, что выход из цикла

произойдёт или при $i>n$ (положительного элемента нет), $K = 0$, или при $i>n$ и $K = 1$ (этот элемент – последний в массиве), или при $i<n$ и $K = 1$ (положительный элемент – a_i, где $1 \le i \le n-1$).

Задача 24. Поиск буквы. Пусть w – слово, состоящее из n букв русского алфавита: $w(1)$, $w(2)$, … , $w(n)$; v – буква, относительно которой требуется установить, входит ли она в слово. Если входит, то укажите номер позиции первого вхождения буквы в слово. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 24).

Будем просматривать последовательно все буквы данного слова. Для реализации этого используем цикл с постусловием. Если буква v не входит в слово, то выход из цикла осуществляется при $i>n$, где i – номер рассматриваемой буквы. Значение переменной K в этом случае равно 0. Если буква v входит в слово, выход из цикла может быть осуществлен раньше, чем i станет больше n, при $K<>0$, где K – номер позиции первого вхождения буквы v в слово w.

Задача 25. Количество положительных элементов. Дана последовательность чисел a_1, a_2, … , a_n. Присвойте переменной m номер K-го положительного элемента этой последовательности. Если в последовательности положительных элементов меньше K, то переменной m присвойте значение 0. Составьте блок-схему алгоритма решения поставленной задачи.

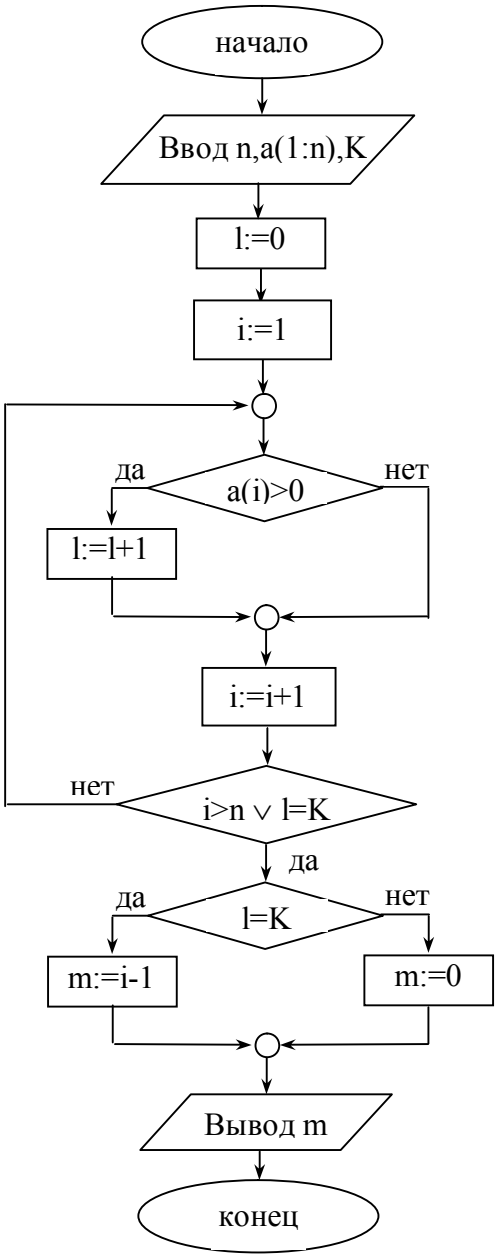
Решение. Смотри блок-схему алгоритма (задача 25).

Подсчет количества положительных элементов массива организуем с помощью цикла с постусловием. Если положительных элементов нет или их меньше, чем K, выход из цикла осуществим при $i>n$, m присвоим значение 0. При $l = K$, где l – число положительных элементов, также реализуем выход из цикла. При этом i получит уже следующее значение. Значит, номер K-го положительного элемента на единицу меньше i.

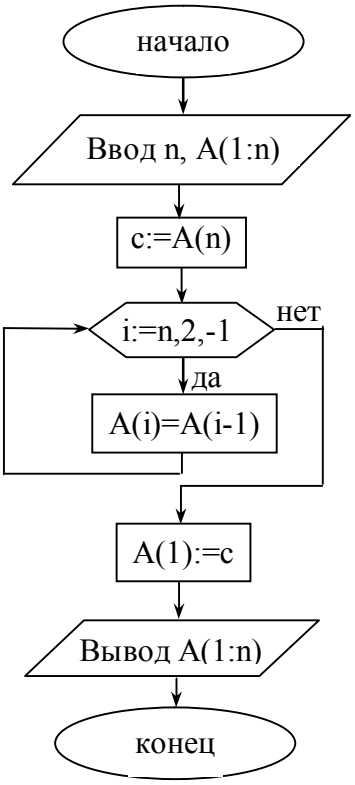
Задача 26. Сдвиг в массиве**.** Переставьте вещественные элементы массива A(1:n) таким образом, чтобы они шли в следующем порядке: A(n), A(1), A(2), … , A(n-2), A(n-1), т.е. в массиве произведите сдвиг элементов на одну позицию вправо. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схему алгоритма (задача 26).

Блок-схема алгоритма (*задача 25*):



Блок-схема алгоритма (*задача 26*):

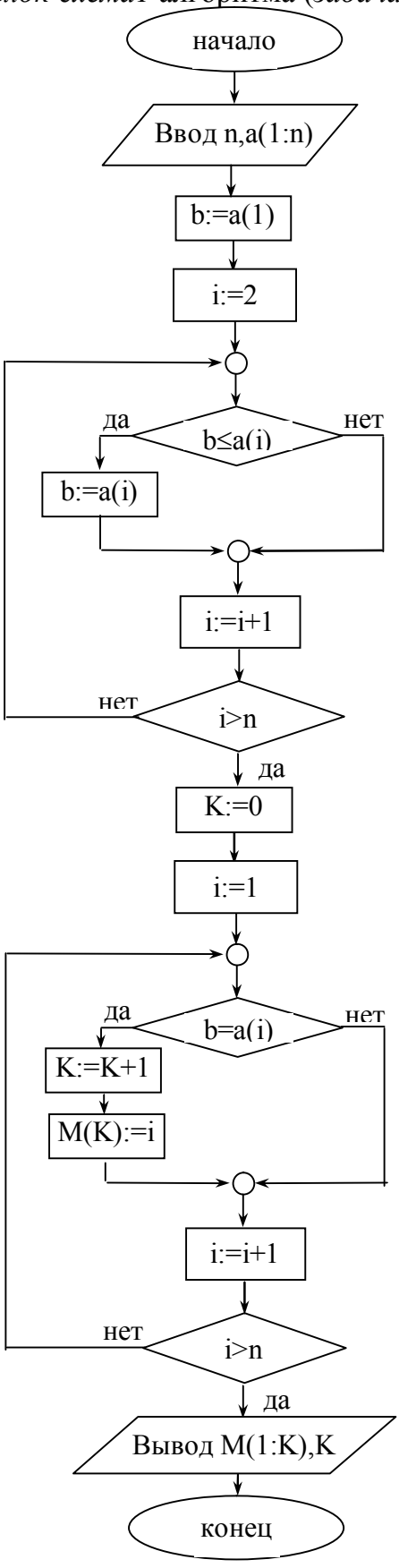


Задача 27. Максимальные элементы массива. Дана последовательность чисел a_1, a_2, … , a_n. Найдите количество максимальных элементов последовательности и их номера. Составьте блок-схему решения поставленной задачи.

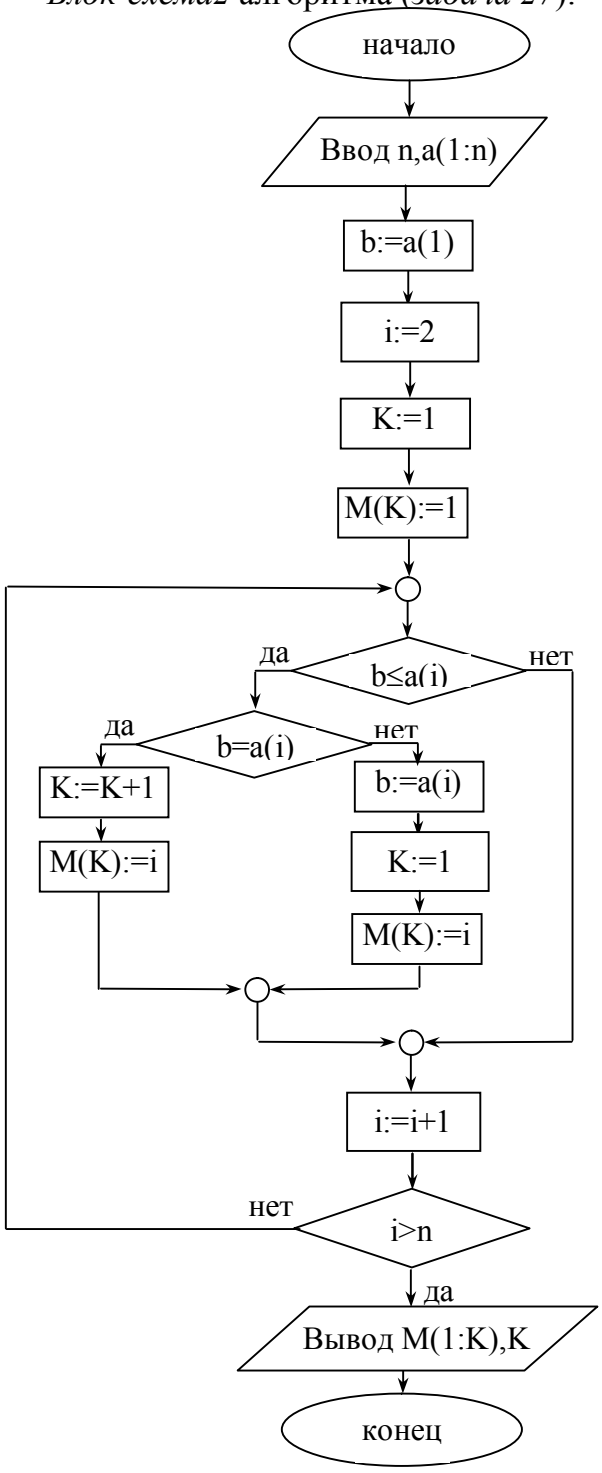
Решение. Смотри блок-схемы 1-2 алгоритма (задача 27).

В блок-схеме1 алгоритма (задача 27) первый цикл организуем для определения максимального элемента, значение которого присваивается переменной b. Начальное значение b положим равным a_1. Второй цикл считает K – количество одинаковых максимальных элементов. Переменная M(K) принимает значение индекса встретившегося максимального элемента. M(K) – числовая последовательность, членами которой являются эти индексы.

Блок-схема1 алгоритма (*задача 27*):



Блок-схема2 алгоритма (*задача 27*):

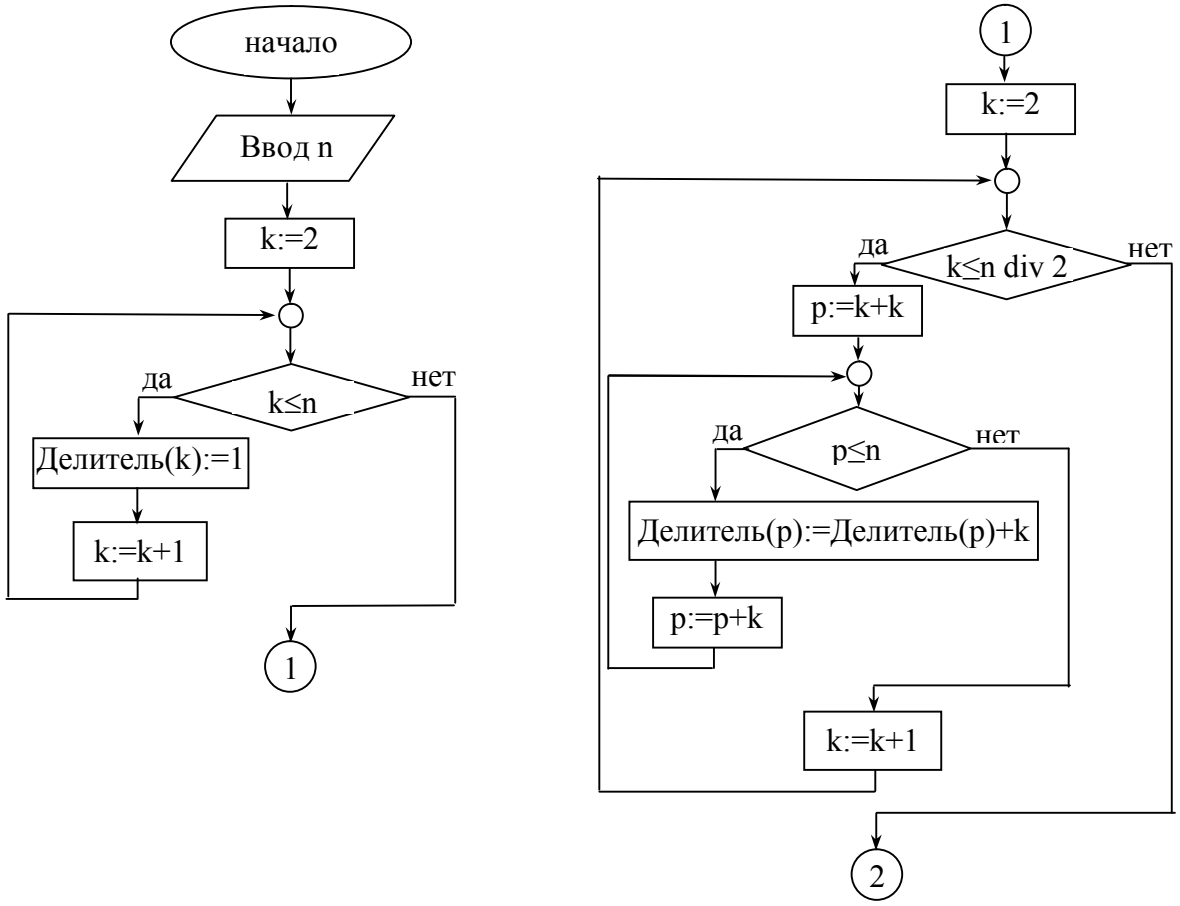


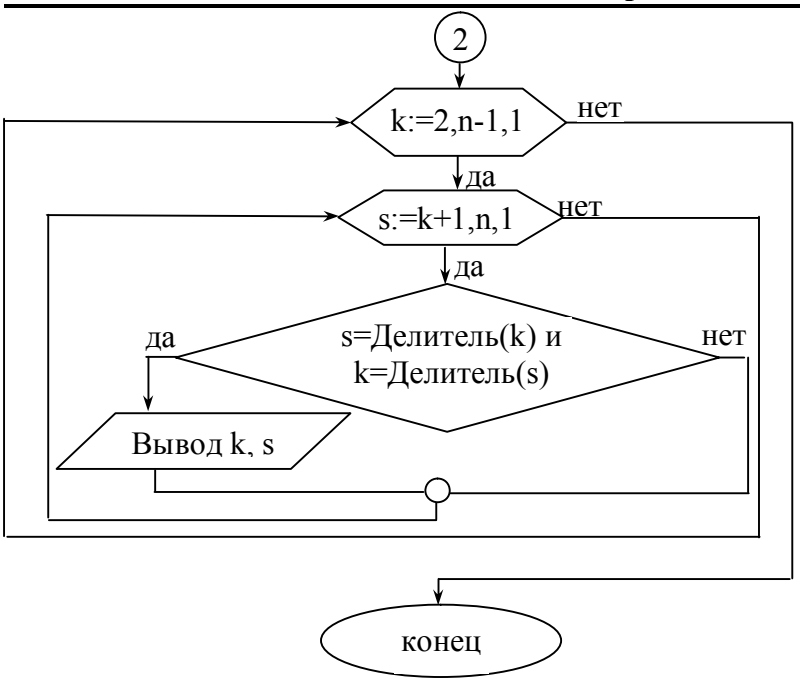
Блок-схема2 алгоритма (задача 27) предполагает в одном цикле на *i*-ом шаге определение максимального элемента из *i* рассмотренных, подсчет количества таких элементов и фиксирование их номеров. Если на (*i*+1)-ом шаге встречается больший элемент, то операторы $b:=a_i$, $K:=1$, $m_K:=i$ зададут новые начальные значения переменных *b*, *K*, m_K, и все вычисления будут проведены относительно нового большего элемента.

Задача 28. Дружественные числа. Дружественными числами называются два натуральных числа, таких, что каждое из них равно сумме всех натуральных делителей другого, исключая само это другое число. Числа 220 и 284 являются дружественными числами, так как сумма делителей числа 220 – это 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284, а сумма делителей числа 284 – это 1 + 2 + 4 + 71 + 142 = 220. Составьте блок-схему алгоритма нахождения дружественных чисел на отрезке [2;*n*].

Решение. Смотри блок-схему алгоритма (задача 28).

Блок-схема алгоритма (*задача 28*):

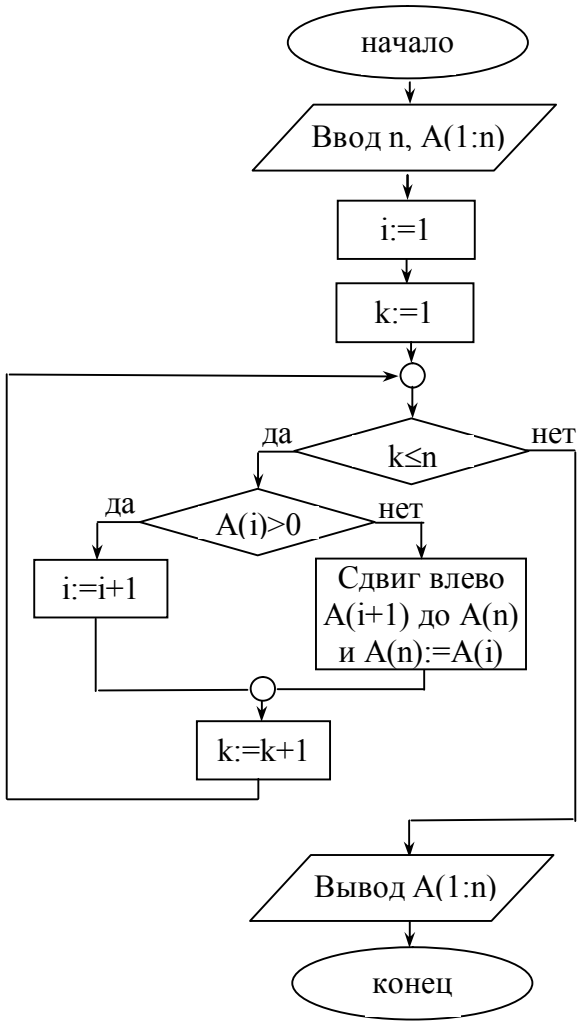




Суммы делителей чисел от 2 до *n* организуем в виде массива Делитель(2:*n*), где Делитель(*k*) – текущая сумма делителей числа *k*. Вычислим суммы делителей всех чисел от 2 до *n*, затем попарно сравним эти суммы, если суммы делителей чисел *k* и *s*, принадлежащих [2;*n*], равны, то числа *k* и *s* – дружественные, их необходимо вывести.

Некоторые пары дружественных чисел: 220 и 284, 1184 и 1210, 2620 и 2924, 5020 и 5564, 6232 и 6368 и т.д.

Блок-схема1 алгоритма (*задача 29*):



Задача 29. Группировка. Дан массив A(1:*n*), элементы которого отличны от нуля. Расположите их в таком порядке, чтобы первыми были все положительные элементы, а затем – все отрицательные, причем порядок следования как положительных, так и отрицательных элементов должен сохраняться. При решении задачи нельзя заводить новую таблицу. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схемы 1-2 алгоритма (задача 29).

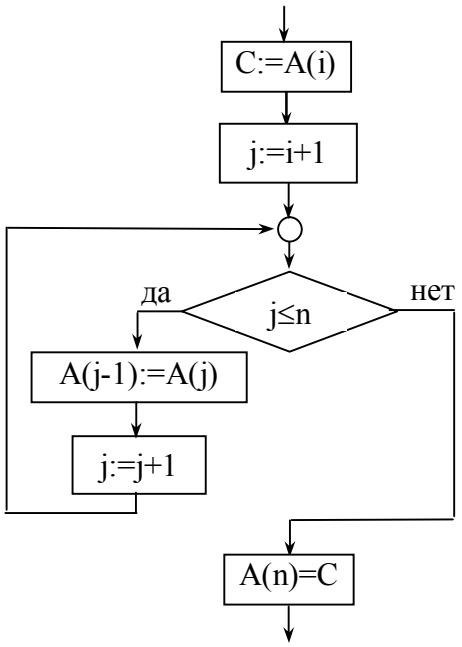
Для решения поставленной задачи будем перебирать элементы мас-

сива с первого по *n*-ый. Если A(*i*)>0, никаких изменений в массиве не производим, увеличиваем *i* на единицу и переходим к исследованию следующего элемента. Если же A(*i*)<0, поставим его на *n*-ое место (в конец таблицы), предварительно освободив это *n*-ое место, произведя сдвиг элементов массива от A(*i*+1) до A(*n*) на один номер влево. Количество просмотренных элементов массива будем запоминать в переменной *k*.

Детализируем блок «сдвиг влево на одну позицию и перестановку A(*i*) на *n*-ое место»: скопируем A(*i*) в переменную *C*, произведем сдвиг и A(*n*) присвоим значение *C*. Смотри блок-схему2 алгоритма (задача 29).

Теперь детализированный блок можно подставить в блок-схему1 алгоритма (задача 29).

Блок-схема2 алгоритма (*задача 29*):



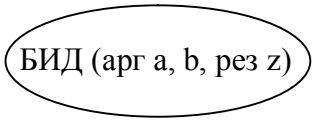
3.4. Блок-схемы алгоритмов, содержащих команды обращения к вспомогательным алгоритмам

Часто при построении алгоритмов решения задач возникает необходимость организации в различных местах алгоритма одной и той же последовательности команд, которая выполняется каждый раз при различных наборах операндов. В таких случаях эти последовательности команд обычно оформляют специальным образом в виде вспомогательных (подчиненных) алгоритмов. А в главном (основном) алгоритме вместо этой последовательности команд записывают одну команду обращения к вспомогательному алгоритму. Алгоритмы, целиком включаемые в состав разрабатываемого алгоритма, называются вспомогательными (подчиненными).

Использование вспомогательных алгоритмов при проектировании алгоритма решения поставленной задачи позволяет целенаправленно идти к решению задачи, использовать при решении разработанные ранее алгоритмы. Кроме того, это позволяет избежать при за-

писи основного алгоритма повторения текста в тех его частях, которые описаны во вспомогательном алгоритме. Использование вспомогательных алгоритмов – средство экономии памяти компьютера: каждый вспомогательный алгоритм существует в одном экземпляре, в то время, как обращаться к нему можно многократно из разных точек программы. При вызове вспомогательного алгоритма активизируется последовательность образующих его операторов, а с помощью передаваемых вспомогательному алгоритму параметров, нужным образом модифицируется реализуемый в нем алгоритм.

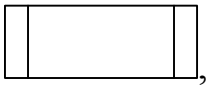
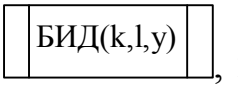
Использование вспомогательных алгоритмов вызывает необходимость оформлять их специальным образом, чтобы иметь возможность в дальнейшем ссылаться на них в основном алгоритме. Формальные способы оформления таких алгоритмов широко применяются в языках программирования, а сами вспомогательные алгоритмы, записанные на языках программирования, называют подпрограммами или процедурами. При использовании блок-схем полная формализация в оформлении вспомогательных алгоритмов, как правило, не производится. Однако, будем придерживаться некоторых принципов оформления. В заголовке вспомогательного алгоритма (блок «начало») за именем вспомогательного алгоритма будем указывать в круглых скобках список так называемых формальных параметров. В списке формальных параметров укажем имена входных и выходных данных (аргументов (арг) и результатов (рез)) алгоритма. Это необходимо для того, чтобы при ссылке на подчиненный алгоритм в основном алгоритме можно было задать значения аргументов, а после исполнения вспомогательного алгоритма, воспользоваться результатами – значениями соответствующих данных. Например, если алгоритм нахождения большего из двух чисел необходимо оформить как вспомогательный, где *a* и *b* – аргументы, а *z* – результат, то блок начала этого алгоритма оформим так:



БИД – имя алгоритма (большее из двух).

Ссылка на вспомогательный алгоритм в основном алгоритме осуществляется с помощью специальной команды вызова вспомогательного алгоритма, в которой указывается имя вспомогательного алгоритма и список фактических параметров, которые должны быть подставлены вместо формальных параметров при исполнении вспомогательного алгоритма. Команда вызова вспомогательного алгоритма имеет вид:

<имя вспомогательного алгоритма> (<список фактических параметров>).

На блок-схеме вызов вспомогательного алгоритма изобразим в виде геометрической фигуры  , внутри которой запишем команду вызова вспомогательного алгоритма. Например, если в основном алгоритме необходимо найти большее из данных k и l, где переменной k было присвоено значение 45, а переменной l – значение 21, и результат присвоить переменной y, то обращение к вспомогательному алгоритму запишем так: 

, где k, l, y – фактические параметры.

Исполнение команды вызова вспомогательного алгоритма эквивалентно исполнению вспомогательного алгоритма с фактическими параметрами. Команда вызова вспомогательного алгоритма исполняется как один шаг основного алгоритма.

Исполнение команды обращения к вспомогательному алгоритму разбивается *на следующие этапы*:

1. На время исполнения команды обращения к вспомогательному алгоритму выполнение основного алгоритма приостанавливается.

2. Формальным параметрам-аргументам вспомогательного алгоритма присваиваются значения фактических параметров, записанных в команде вызова вспомогательного алгоритма.

3. Исполняется вспомогательный алгоритм с полученными значениями аргументов.

4. Значения результатов-параметров вспомогательного алгоритма присваиваются соответствующим значениям параметров, записанных в команде вызова вспомогательного алгоритма.

5. Выполняется команда основного алгоритма, следующая за командой вызова вспомогательного алгоритма.

Замечание: Значения данных вспомогательного алгоритма, не переданные в основной алгоритм после исполнения вспомогательного алгоритма, становятся недоступными. Количество, порядок, тип параметров в команде вызова вспомогательного алгоритма должны соответствовать количеству, порядку, типу параметров в заголовке вспомогательного алгоритма.

Схематично передачу значений параметров при исполнении команды обращения к вспомогательному алгоритму можно изобразить так:

Задача 30. Наибольший общий делитель. Составьте блок-схему алгоритма нахождения z – наибольшего общего делителя четырех натуральных чисел a, b, c, d, используйте для решения задачи вспомогательный алгоритм нахождения наибольшего общего делителя t двух натуральных чисел m и n.

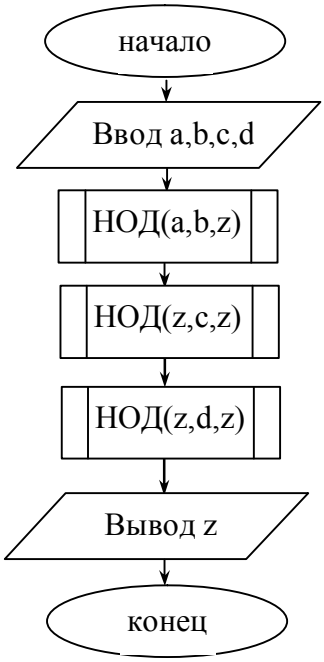
Решение. Смотри блок-схемы 1-2 алгоритма (задача 30).

Составим блок-схему вспомогательного алгоритма НОД, нахождения наибольшего общего делителя двух натуральных чисел m и n, значение наибольшего общего делителя двух натуральных чисел присвоим переменной t. Для решения поставленной задачи воспользуемся тремя предложениями (обозначим НОД (m,n) – наибольший общий делитель m и n):

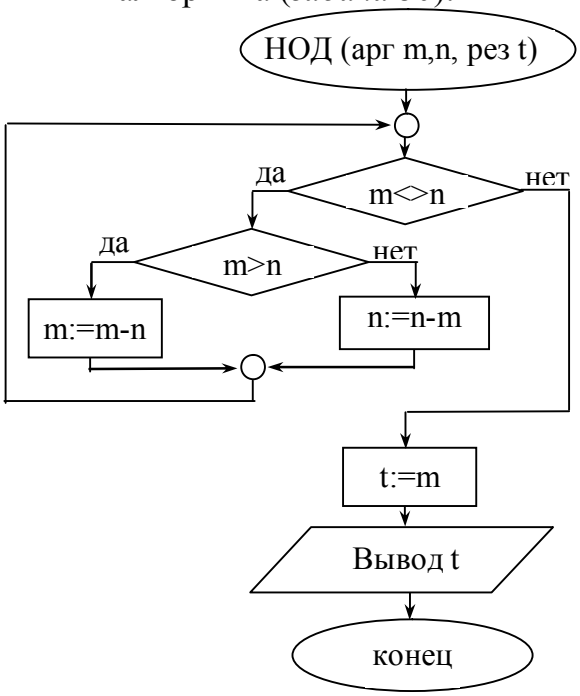
1. НОД(m,n) = НОД(m-n,n), если m>n.
2. НОД(m,n) = НОД(m,n-m), если n>m.
3. НОД(m,n) = m, если m = n.

НОД(25,15) = НОД(10,15) (1);
НОД(10,15) = НОД(10,5) (2);
НОД(10,5) = НОД(5,5) (1);
НОД(5,5) = 5 (3) => НОД(25,15) = 5.

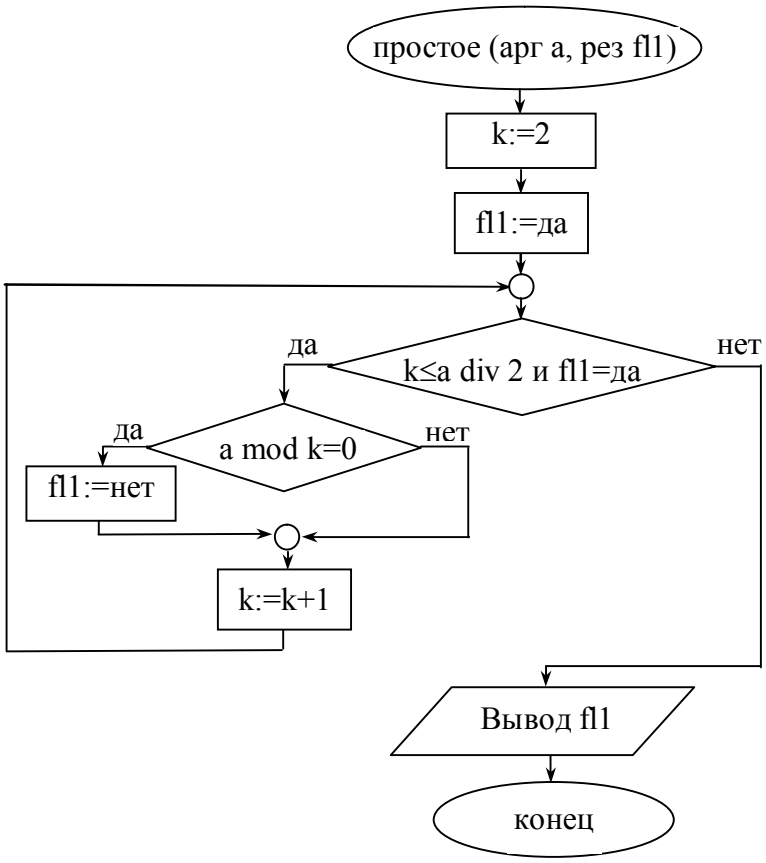
Блок-схема1 основного алгоритма (*задача 30*):



Блок-схема2 вспомогательного алгоритма (*задача 30*):



Блок-схема1 вспомогательного алгоритма (*задача 31*)

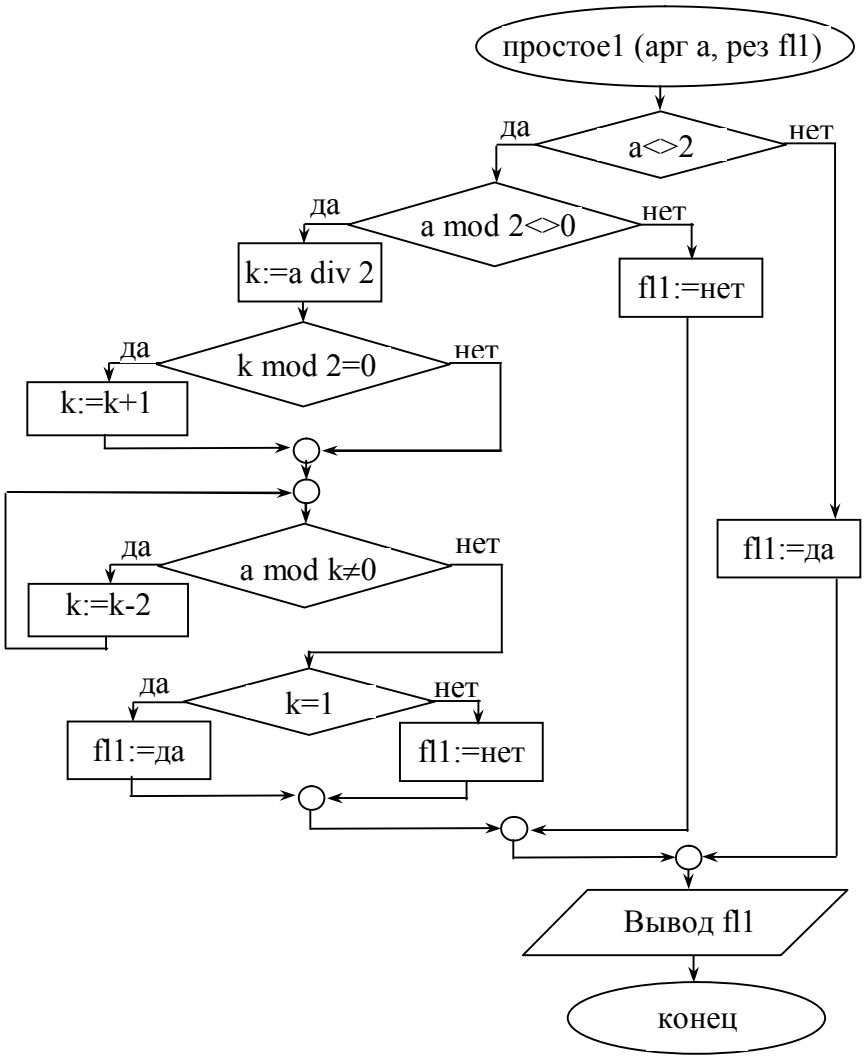


Задача 31. Числа Мерсенна. Числа вида 2^p-1, где p – простое число, называются числами Мерсенна[7]. Многие числа Мерсенна также являются простыми. Составьте алгоритм нахождения всех чисел Мерсенна, принадлежащих отрезку [2;n].

Решение. Смотри блок-схемы 1-3 алгоритма (задача 31).

[7] М. Мерсенн (Marin Mersenne, 1588-1648) – французский математик.

Блок-схема2 вспомогательного алгоритма (*задача 31*):



Для решения задачи будем получать числа вида 2^p-1, где p = 2, 3, … . Если p – простое, то 2^p-1 – число Мерсенна. Рассмотрим, как можно получить следующего кандидата в числа Мерсенна, если известен предыдущий проверенный кандидат:

Кандидат = 2^p-1,

где p = 2, 3, 4, 5, 6, …;

следующего кандидата обозначим Кандидат1:

Кандидат1 = 2^{p+1}-1;

2^{p+1}-1 = 2∗2^p-1 = 2∗(2^p-1) + 1; Кандидат1 = 2 ∗ Кандидат + 1.

Нового кандидата в числа Мерсенна будем проверять осторожно, чтобы не было превышено n. Если проверяемых кандидатов на отрезке [2;n] больше нет, то флажку *fl* будем присваивать значение «да». До проверки первого кандидата флажку *fl* присвоим значение «нет». Найдем $n1 = n/2$:

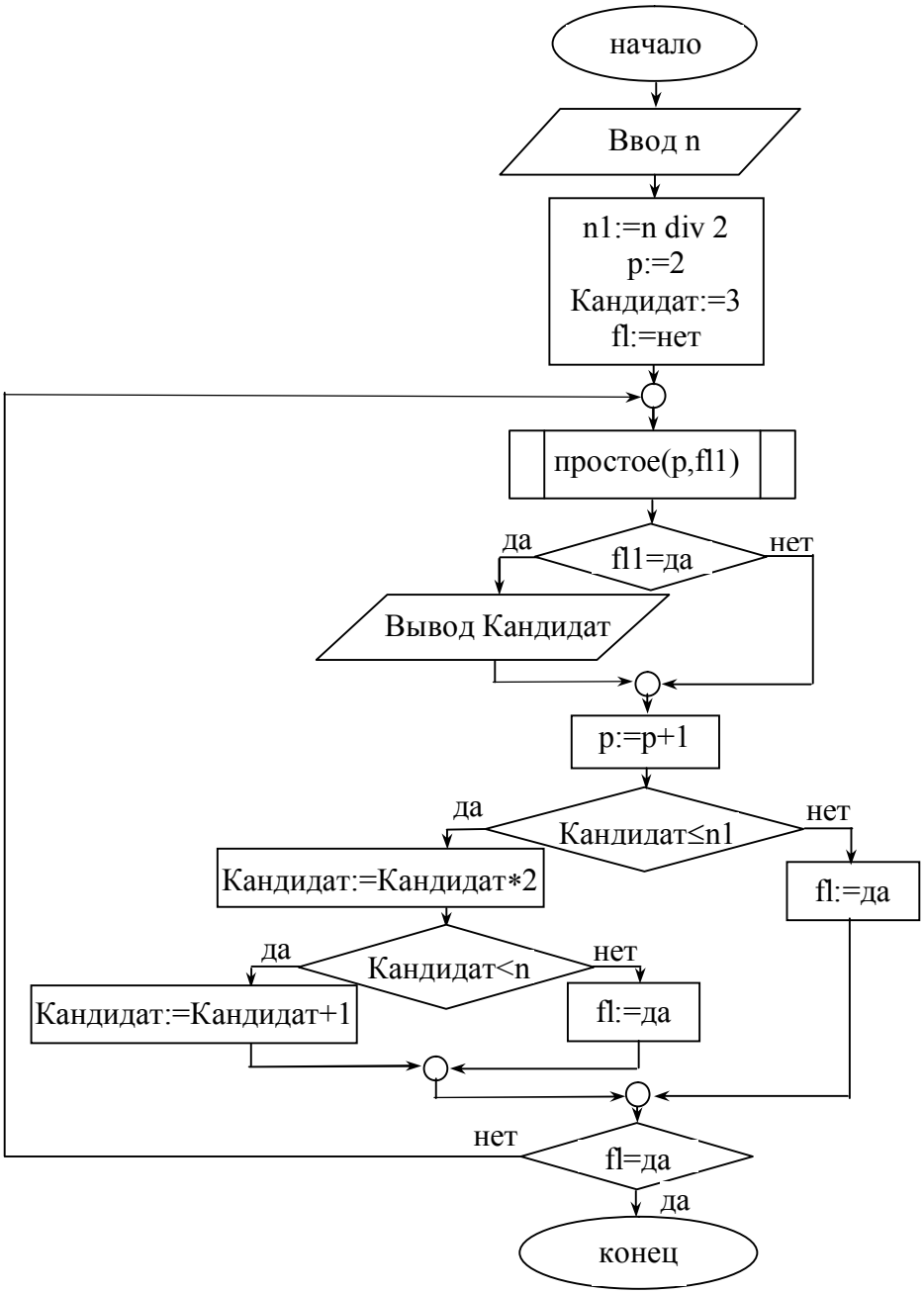
если Кандидат≤n1, то Кандидат1:=2∗Кандидат, иначе fl:=да;

если Кандидат1<n, то Кандидат1:=Кандидат1+1, иначе fl:=да.

Легко проверить, что число 2^2-1, равное 3 – число Мерсенна. Проверку высказывания «p – число простое» оформим в виде одного из вспомогательных алгоритмов *простое (арг a, рез fl1)* или *простое1 (арг a, рез fl1).*

Составим блок-схему вспомогательного алгоритма *простое1 (арг a, рез fl1)*, алгоритма проверки является ли число a простым числом. Заметим, что все четные числа, большие 2 являются составными. Проверим, имеет ли число a (нечетное, большее 2) делители, отличные от 1 и самого себя. Исследуем числа от 2 до [a/2]+1, где [a/2] – целая часть частного a/2 (a>0). На отрезке [[a/2]+1;a-1] не

Блок-схема3 основного алгоритма (*задача 31*):



могут находиться делители числа *a*. Делителями нечетного числа *a* могут быть только нечетные числа *k*, меньшие [*a*/2]+1. Будем их получать так:

k:=[a/2]+1-2;
k:=k-2

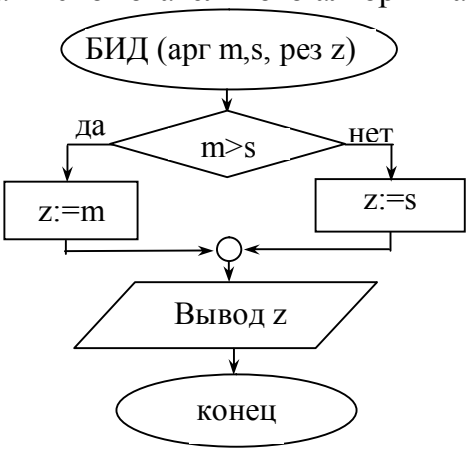
и т.д. до тех пор, пока *k*<>1. Проверку реализуем в виде цикла. Из цикла выйдем, если *k* является делителем числа *a* (*k*=1 или *k*<>1). По окончании цикла проверим значение *k*: если *k*=1, то число простое и *fl*1:=да, иначе число составное и *fl*1:=нет.

Числа Мерсенна: 3, 7, 31, 127, 2047, 8191, 131071, 524287, 8388607, 536870911, 2147483647, …

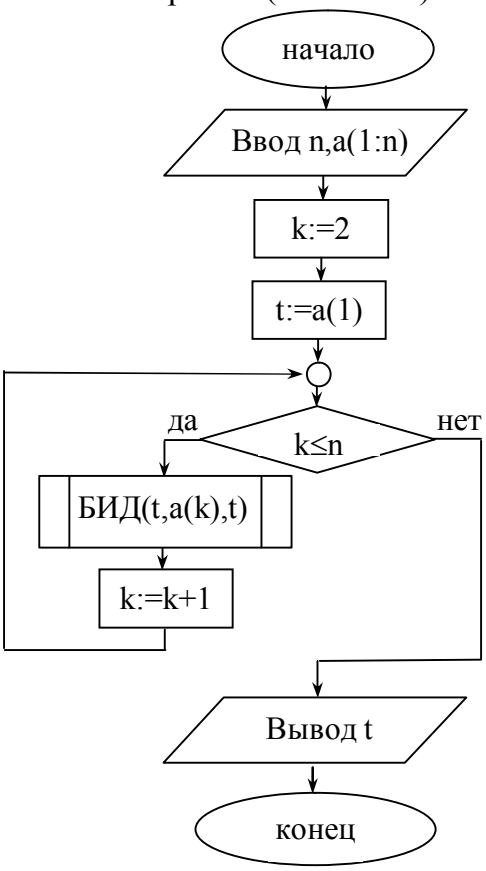
Задача 32. Наибольший элемент в массиве. Составьте алгоритм поиска наибольшего элемента *t* в линейной таблице из *n* чисел a(1:*n*), используя алгоритм БИД (нахождения большего из двух чисел), как вспомогательный.

Решение. Смотри блок-схемы 1-2 алгоритма (задача 32).

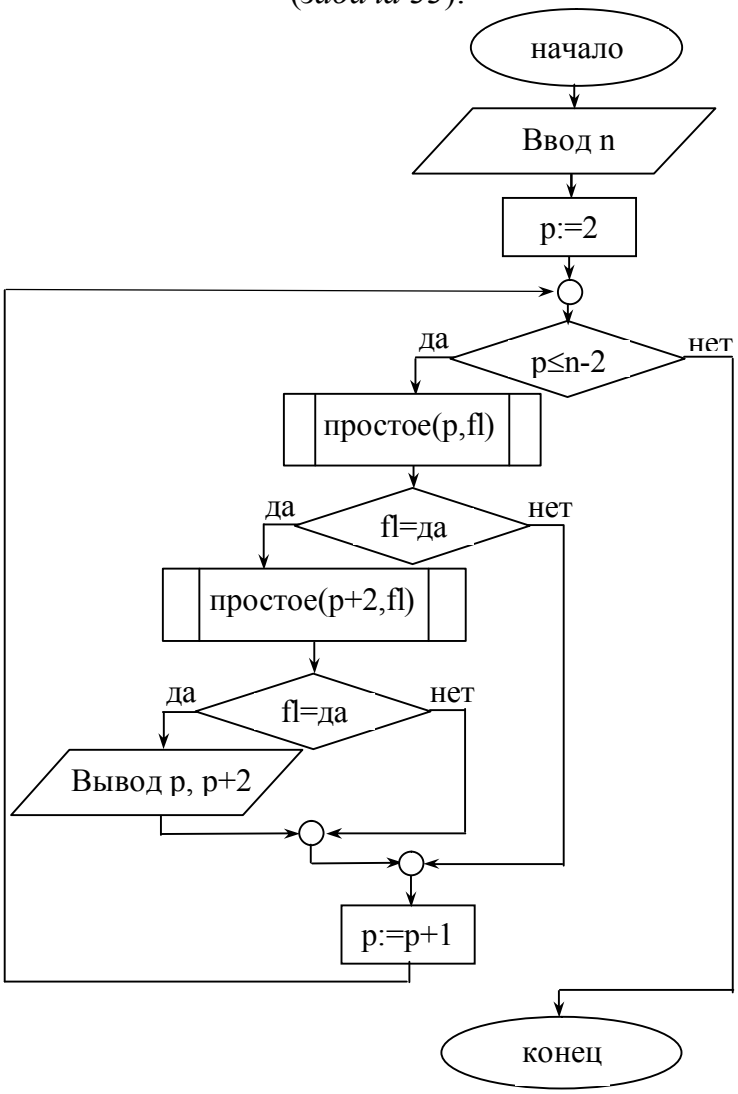
Блок-схема1 вспомогательного алгоритма (*задача 32*):



Блок-схема2 основного алгоритма (*задача 32*):



Блок-схема алгоритма (*задача 33*):



Задача 33. Близнецы. Два нечетных простых числа, отличающихся на два, называются близнецами. Близнецами являются, например, числа 5 и 7, 11 и 13, 17 и 19 и т.д. Составьте блок-схему нахождения близнецов на отрезке от 2 до *n*.

Решение. Смотри блок-схему алгоритма (задача 33).

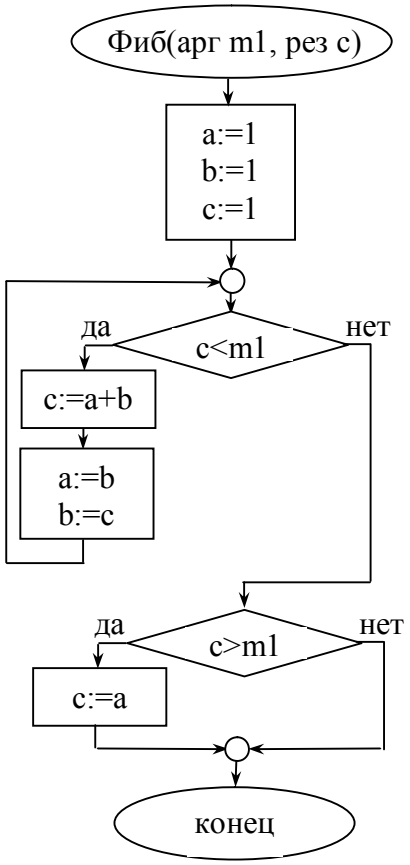
Будем определять является ли число $p \in [2;n]$ простым, если p – простое, то проверяем простое ли число $p+2$, если p и $p+2$ простые, то их выводим; далее исследуем числа $p+1$ и $p+3$ и т.д. Для определения является ли число p простым воспользуемся вспомогательным алгоритмом *простое (арг a, рез fl1)* или *простое1 (арг a, рез fl1)* (стр. 60-61, задача 31).

Задача 34. Сумма чисел Фибоначчи. Представьте число *m* в виде суммы наибольших чисел Фибоначчи. Составьте блок-схему алгоритма решения поставленной задачи.

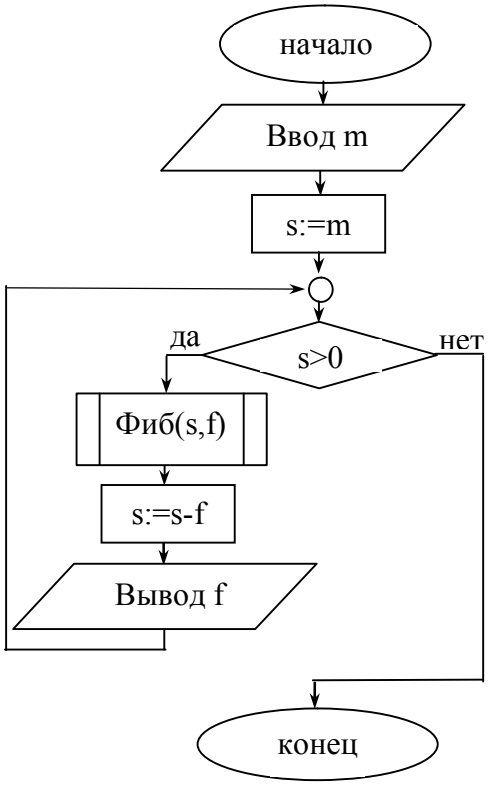
Решение. Смотри блок-схемы 1-2 алгоритма (задача 34).

Для решения поставленной задачи воспользуемся алгоритмом задачи 18, стр. 42, как вспомогательным.

Блок-схема1 вспомогательного алгоритма (*задача 34*):



Блок-схема2 основного алгоритма (*задача 34*):

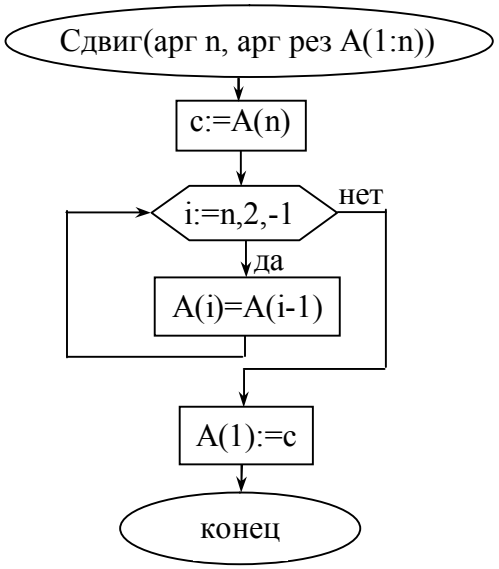


Задача 35. Сдвиг1 элементов в массиве. Составьте алгоритм циклической перестановки элементов массива B(1:*n*) на заданное число *k* шагов так, чтобы элемент B(*i*) переместился на место B(*i*+*k*), а последние *k* элементов массива, которым «не хватило места», переместились, соответственно, на первые *k* мест, освободившихся при сдвиге. (Массив B(1), B(2), … , B(*n*) преобразовался в массив B(*n*-*k*+1), … , B(*n*-1), B(*n*), B(1), B(2), … , B(*n*-*k*).) Составьте блок-схему алгоритма решения поставленной задачи.

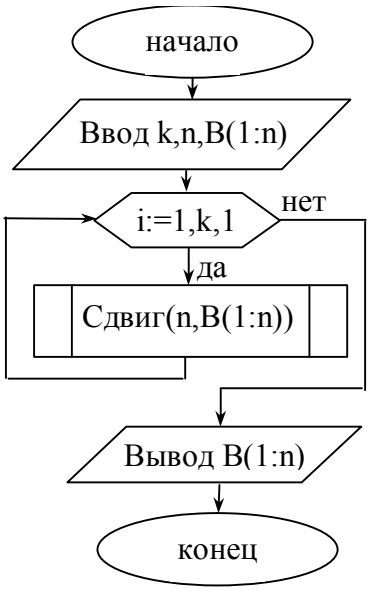
Решение. Смотри блок-схемы 1-2 алгоритма (задача 35).

Для решения поставленной задачи воспользуемся составленным алгоритмом задачи 26, стр. 51, как вспомогательным.

Блок-схема1 вспомогательного алгоритма (*задача 35*):



Блок-схема2 основного алгоритма (*задача 35*):



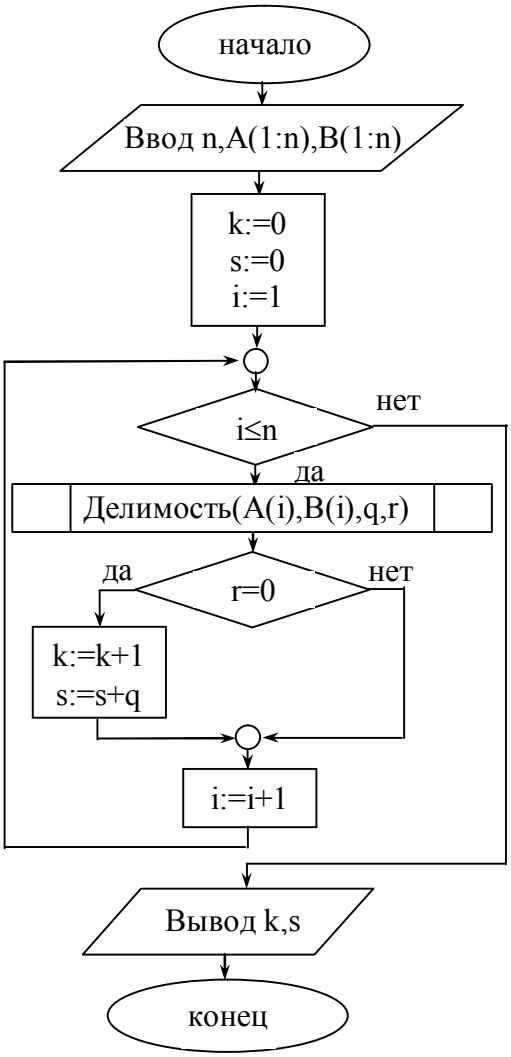
Задача 36. Кратные пары. Заданы два массива A(1:*n*) и B(1:*n*) натуральных чисел. Подсчитайте число *k* пар (a_i,b_i), $1 \le i \le n$, соответствующих друг другу кратных чисел этих массивов и найдите сумму частных от деления большего числа каждой такой пары на меньшее. Составьте блок-схему алгоритма решения поставленной задачи.

Решение. Смотри блок-схемы 1-2 алгоритма (задача 36).

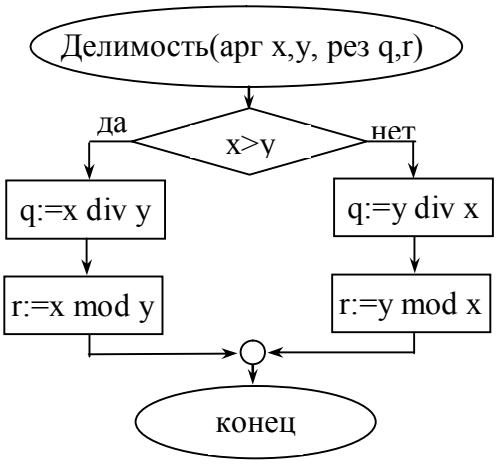
Будем перебирать последовательно все пары соответствующих чисел этих массивов и применим к каждой паре вспомогательный алгоритм Делимость, в

котором для каждой пары чисел массивов определим частное и остаток от деления большего числа на меньшее.

Блок-схема1 основного алгоритма (*задача 36*):



Блок-схема2 вспомогательного алгоритма (*задача 36*):



4. Алгостихи

Для формирования навыков записи алгоритмов с использованием базовых управляющих команд организации действий можно использовать алгоритмические стихи (алгостихи), т.е. стихотворения, по содержанию представляющие некоторый алгоритм.

Рассмотрим пример алгостиха, который можно интерпретировать основными управляющими командами.

Стихотворение *«Царица мух»*, *Н.Заболоцкий*

Если ты, мечтой томим,
Знаешь слово Элоим[8],
Муху странную бери,
Муху в банку посади,
С банкой по полю ходи,
За приметами следи.

Если муха чуть шумит,
Под ногою медь лежит.
Если усиком ведет,
К серебру тебя зовет.
Если хлопает крылом,
Под ногами злата ком.

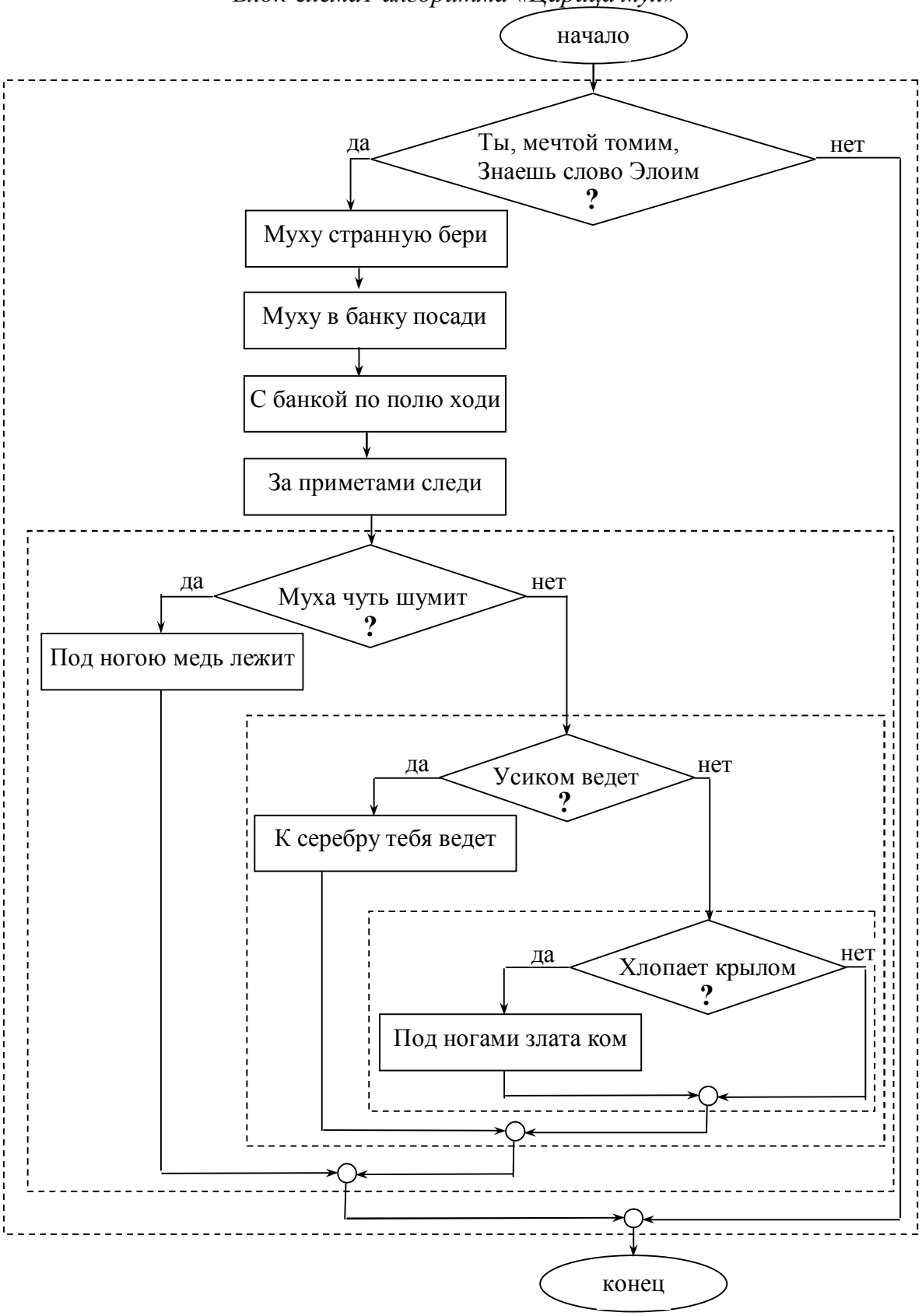
Из истории создания стихотворения «Царица Мух»

Н. Заболоцкий (1903-1958 гг.) так пишет об истории создания этого стихотворения.

Знаменитый ученый Агниппа Ноттенгеймский Генрих Корнелий (1486-1535 гг.) – разносторонний ученый, царицей мух называет какую-то таинственную муху, величиной с крупного шмеля, которая любит садиться на водяное растение, называемое Pluteau plantagine, и с помощью которой индусы якобы отыскивают клады на своей родине. Когда вы имеете в своем распоряжении одну из таких мух – посадите ее в прозрачный ящичек. Её помещение надо освежать 2 раза в день и давать ей растение, на котором ее поймали. Она может жить в таких условиях почти месяц. Чтобы узнать направление скрытых на известной глубине сокровищ, надо, чтобы стояла хорошая установившаяся погода. Тогда, взяв ящичек с мухой, отправляйтесь в путь, постоянно посматривая и подмечая ее движения. Когда вы будете находиться над местом, содержащим золото или серебро, муха замахает крыльями, и чем ближе вы будете, тем сильнее будут ее движения. Если в недрах сокрыты драгоценные камни, вы заметите содрогания в лапках и усиках. В том же случае, если там находятся лишь неблагородные металлы, как медь, железо, свинец и пр., муха будет ходить спокойно, но чем быстрее, тем ближе к поверхности они находятся. Нечто похожее на это предание можно услышать и в русских деревнях.

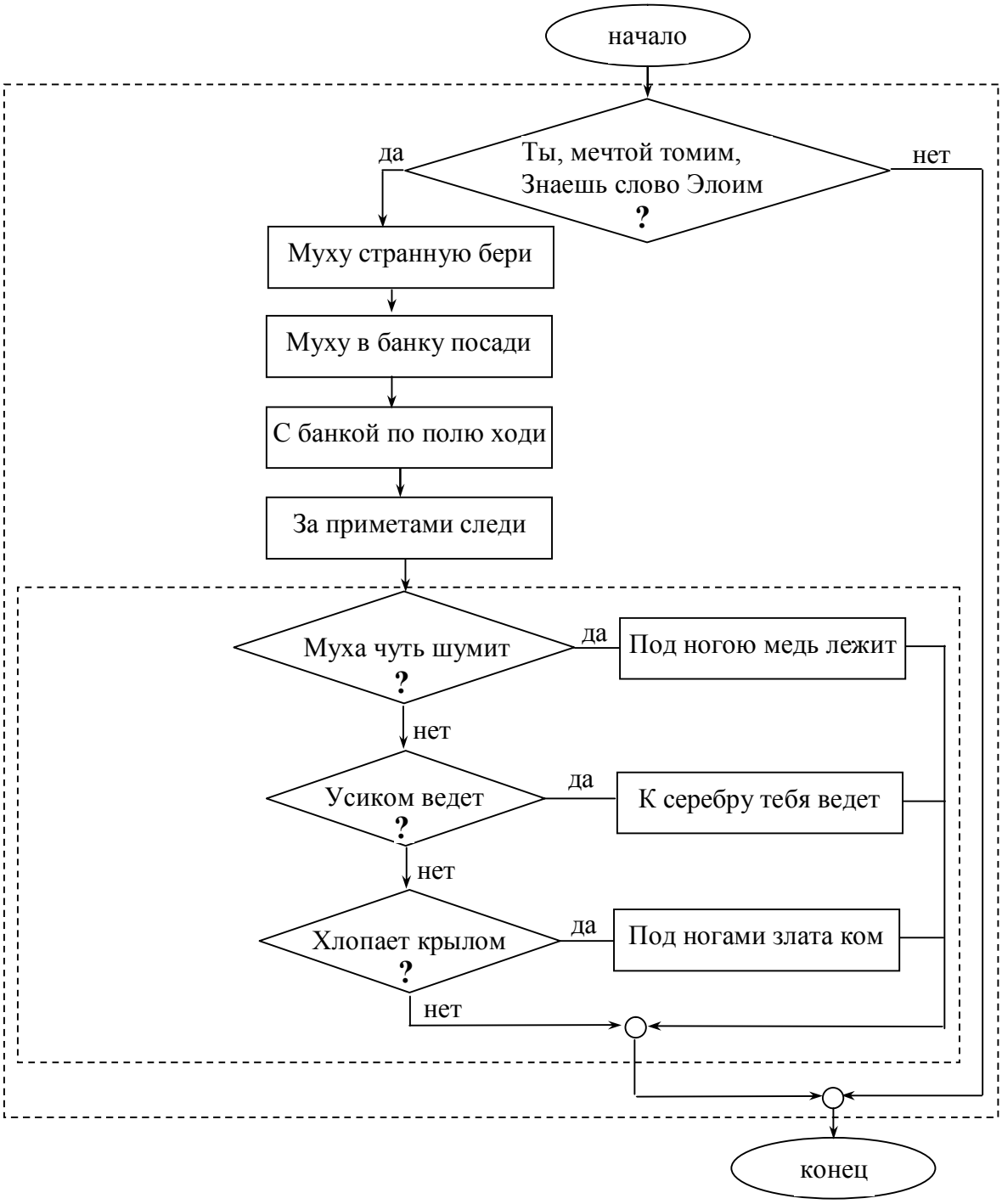
[8] Элоим – Бог-отец, до IX века до нашей эры (античная мифология).

Блок-схема1 алгоритма «Царица мух»



Для составления блок-схемы1 использовано вложенное ветвление.

Блок-схема2 алгоритма «Царица мух»



Для составления блок-схемы2 использована команда выбора в неполной форме.

5. Алгоритмы для исполнителя МНР (машины с неограниченными регистрами)

Опишем исполнителя МНР. МНР состоит из памяти данных и программной памяти. Ячейки памяти данных называют регистрами и обозначают R0, R1, R2 и т.д. до бесконечности. В каждом регистре может содержаться любое целое неотрицательное число. Число, содержащееся в Rn, или значение регистра Rn будем обозначать r_n. В каждый момент времени только конечное число регистров содержит какие-то числа, т.е. память данных является не бесконечной, а потенциально бесконечной. МНР работает по программе. Программой называется пронумерованная конечная последовательность команд, хранящаяся в программной памяти. Программная память МНР также является потенциально бесконечной. Будем считать, что во всех ячейках, свободных от команд программы записана специфическая команда – Stop, результатом исполнения которой является остановка выполнения программы.

Система команд МНР содержит следующие команды:

1. *Команда обнуления*.

Команда обнуления записывается Z(n). Команда Z(n) выполняется так: содержимое регистра Rn обнуляется, т.е. r_n становится равным нулю (r_n:=0), содержимое других регистров не изменяется.

Пример 5. Пусть начальная конфигурация МНР имеет вид:

R0	R1	R2	R3	R4	R5	…
9	6	5	27	7	0	…

МНР выполнит команду обнуления – Z(3). Результатом будет следующая конфигурация:

R0	R1	R2	R3	R4	R5	…
9	6	5	0	7	0	…

(*)

2. *Команда прибавления единицы*.

Команда прибавления единицы имеет вид S(n). Команда S(n) выполняется так: содержимое регистра Rn увеличивается на единицу, т.е. r_n:=r_n+1, новое значение r_n равно старому значению, сложенному с 1, содержимое других регистров не изменяется.

Пример 6. Пусть начальная конфигурация МНР имеет вид (*). МНР выполнит команду S(4). Тогда новая конфигурация имеет вид:

R0	R1	R2	R3	R4	R5	…
9	6	5	0	8	0	…

(**)

3. *Команда переадресации*.

Команда переадресации имеет вид T(*m*,*n*). Команда T(*m*,*n*) выполняется так: содержимое регистра R*n* заменяется числом r_m, содержащимся в регистре R*m*, т.е. $r_n := r_m$, содержимое других регистров (включая R*m*) не изменяется.

Пример 7. Пусть начальная конфигурация МНР имеет вид (**). МНР выполнит команду T(4,2). Тогда новая конфигурация МНР имеет вид:

R0	R1	R2	R3	R4	R5	…
9	6	8	0	8	0	…

В МНР есть особый регистр, который назовем *счетчиком адресов команд* (САК). При запуске программы на выполнение в САК заносится 1. *Выполнение каждой команды МНР состоит из четырех этапов:*

1. Читается адрес из САК.
2. Запоминается команда из памяти по этому адресу.
3. САК увеличивается на единицу.
4. Выполняется запомненная команда.

МНР выполняет каждую команду программы в 4 этапа, пока не встретит команду *Stop*.

Команды обнуления, прибавления единицы и переадресации называются *арифметическими*. После каждой команды типа 1-3 выполняется команда со следующим номером.

4. *Команда условного перехода*.

В работе алгоритма могут быть моменты, когда предписываются альтернативные действия, зависящие от результата работы алгоритма к этому моменту. В других ситуациях может потребоваться повторить группу команд несколько раз. МНР может выполнять такие процедуры, используя команды условного перехода; они позволяют делать скачки вперед и назад в списке команд. Мы будем использовать команду условного перехода для совершения, например, следующего действия: «Если $r_2=r_6$, перейди к десятой команде программы, в про-

тивном случае, перейди к следующей команде программы». Команда, вызывающая это действие, записывается как J(2,6,10).

Команда условного перехода в общем виде записывается так J(*m*,*n*,*q*). Выполнение команды осуществляется следующим образом: сравнивается содержимое регистров R*m* и R*n*, если $r_m=r_n$, то МНР переходит к выполнению *q*-ой команды исполняемой программы, в САК заносится число *q*; если $r_m\neq r_n$, то МНР переходит к выполнению команды, следующей в программе за J(*m*,*n*,*q*), САК увеличивается на единицу, содержимое регистров не изменяется. Если условный переход невозможен ввиду того, что в исполняемой программе команд меньше, чем *q*, то МНР прекращает работу.

С помощью данной команды можно осуществлять и *безусловный переход*. Для этого команду нужно записать следующим образом: J(*m*,*m*,*q*).

Работа на МНР состоит из следующих этапов:

1. Заполняем регистры памяти данных какими-то числами, т.е. задаем так называемую *начальную конфигурацию*.

2. Заполняем командами ячейки программной памяти.

3. Записываем в САК единицу, запускаем МНР и она работает так, как было описано выше до тех пор, пока не встретит команду Stop. При этом полученное по окончании работы машины содержимое регистров памяти данных называется *конечной конфигурацией*.

Примеры алгоритмов

Задача 37. Напишите для исполнителя МНР алгоритм вычисления суммы целых положительных чисел *x* и *y*, результат запишите в регистр R0.

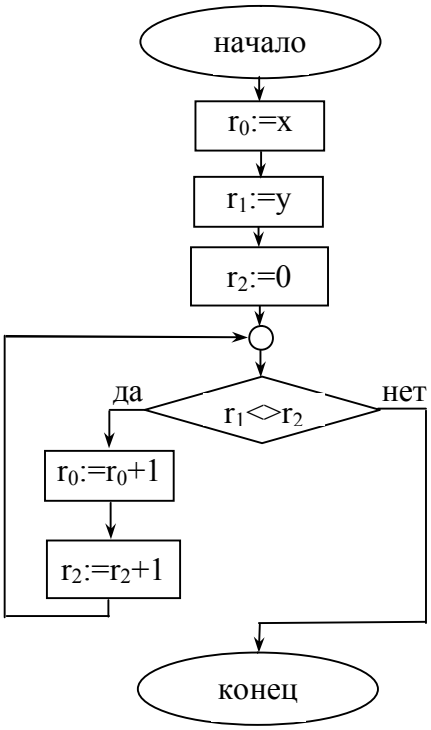
Решение. Поместим *x* в регистр R0, *y* – в регистр R1, т.е. зададим начальную конфигурацию:

R0	R1	…
x	*y*	…

Конечная конфигурация:

R0	…
x+*y*	…

Блок-схема алгоритма:



Алгоритм вычисления *x+y*:

1. Z(2)
2. J(1,2,6)
3. S(0)
4. S(2)
5. J(0,0,2)
6. Stop

Исполнение алгоритма при x=3, y=2

Исполнение алгоритма представим в виде последовательности состояний конфигурации МНР после исполнения очередной команды.

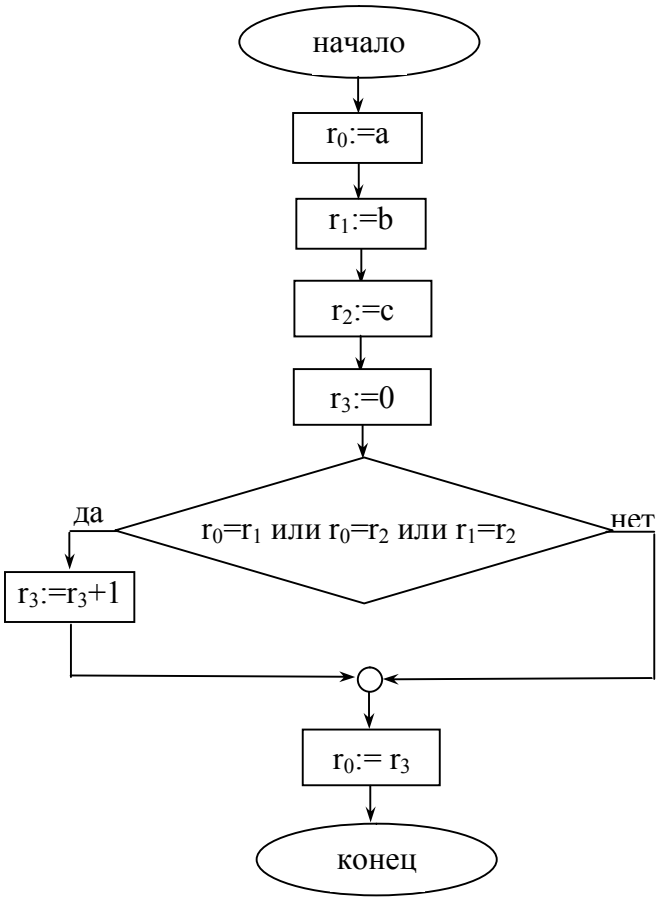
Шаги	САК	Исполняемая команда	Конфигурация МНР после исполнения команды				Проверяемое условие
			R0	R1	R2	R3	
0			3	2	…	…	
1	1	Z(2)	3	2	0	…	
2	2	J(1,2,6)	3	2	0	…	2=0 (нет)
3	3	S(0)	4	2	0	…	
4	4	S(2)	4	2	1	…	
5	5	J(0,0,2)	4	2	1	…	4=4 (да) САК=2
6	2	J(1,2,6)	4	2	1	…	2=1 (нет)
7	3	S(0)	5	2	1	…	
8	4	S(2)	5	2	2	…	
9	5	J(0,0,2)	5	2	2	…	5=5 (да) САК=2
10	2	J(1,2,6)	5	2	2	…	2=2 (да) САК=6
11	6	Stop	5	2	2		

Задача 38. Напишите для исполнителя МНР алгоритм определения является ли треугольник с заданными сторонами *a*, *b*, *c* равнобедренным: $t = \begin{cases} 1, \text{если } a = b \text{ или } a = c \text{ или } b = c; \\ 0, \text{в противном случае} \end{cases}$. Результат запишите в регистр R0.

Решение. Зададим начальную конфигурацию:

R0	R1	R2	…
a	*b*	*c*	…

Блок-схема алгоритма:



Алгоритм вычисления *t*:

1. Z(3)
2. J(0,1,6)
3. J(0,2,6)
4. J(1,2,6)
5. J(0,0,7)
6. S(3)
7. T(3,0)
8. Stop

Конечная конфигурация:

R0	…
t	…

Исполнение алгоритма при a=5, b=4, c=4

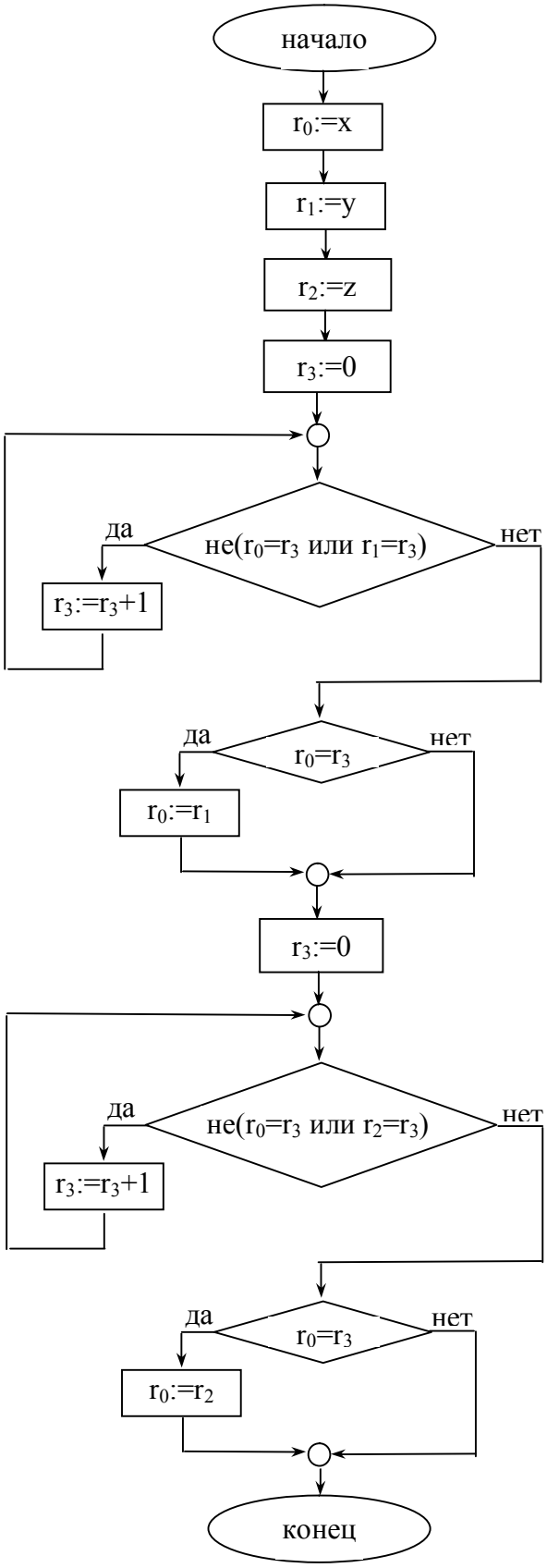
Шаги	САК	Исполняемая команда	Конфигурация МНР после исполнения команды				Проверяемое условие
			R0	R1	R2	R3	
0			5	4	4	…	
1	1	Z(3)	5	4	4	0	
2	2	J(0,1,6)	5	4	4	0	5=4 (нет)
3	3	J(0,2,6)	5	4	4	0	5=4 (нет)
4	4	J(1,2,6)	5	4	4	0	4=4 (да) САК=6
5	6	S(3)	5	4	4	1	
6	7	T(3,0)	1	4	4	1	
7	8	Stop	1	4	4	1	

Задача 39. Напишите для исполнителя МНР алгоритм нахождения большего из трех целых положительных чисел $t = \max(x,y,z)$. Результат запишите в регистр R0.

Решение. Зададим начальную конфигурацию:

R0	R1	R2	…
x	*y*	*z*	…

Блок-схема алгоритма:



В регистре R3 будем вначале хранить min(*x*,*y*), а затем min(max(*x*,*y*),*z*)).

Алгоритм вычисления max(*x*,*y*,*z*):

1. Z(3)
2. J(0,3,6)
3. J(1,3,7)
4. S(3)
5. J(0,0,2)
6. T(1,0)
7. Z(3)
8. J(0,3,12)
9. J(2,3,13)
10. S(3)
11. J(0,0,8)
12. T(2,0)
13. Stop

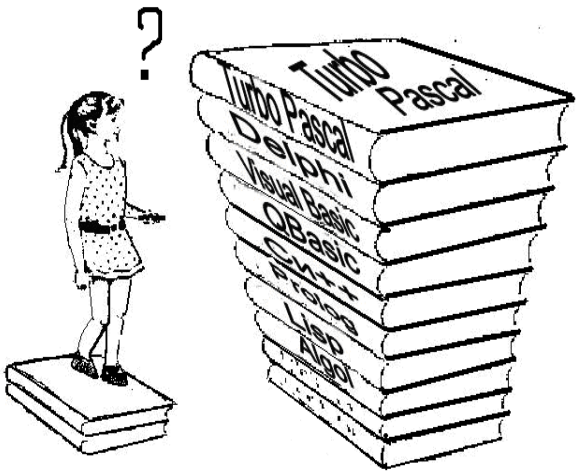
Конечная конфигурация:

R0	…
max(*x*,*y*,*z*)	…

Исполнение алгоритма при x=3, y=2, z=4

Шаги	САК	Исполняемая команда	Конфигурация МНР после исполнения команды				Проверяемое условие
			R0	R1	R2	R3	
0			3	2	4	…	
1	1	Z(3)	3	2	4	0	
2	2	J(0,3,6)	3	2	4	0	3=0 (нет)
3	3	J(1,3,7)	3	2	4	0	2=0 (нет)
4	4	S(3)	3	2	4	1	
5	5	J(0,0,2)	3	2	4	1	3=3 (да) САК=2
6	2	J(0,3,6)	3	2	4	1	3=1 (нет)
7	3	J(1,3,7)	3	2	4	1	2=1 (нет)
8	4	S(3)	3	2	4	2	
9	5	J(0,0,2)	3	2	4	2	3=3 (да) САК=2
10	2	J(0,3,6)	3	2	4	2	3=2 (нет)
11	3	J(1,3,7)	3	2	4	2	2=2 (да) САК=7
12	7	Z(3)	3	2	4	0	
13	8	J(0,3,12)	3	2	4	0	3=0 (нет)
14	9	J(2,3,13)	3	2	4	0	4=0 (нет)
15	10	S(3)	3	2	4	1	
16	11	J(0,0,8)	3	2	4	1	3=3 (да) САК=8
17	8	J(0,3,12)	3	2	4	1	3=1 (нет)
18	9	J(2,3,13)	3	2	4	1	4=1 (нет)
19	10	S(3)	3	2	4	2	
20	11	J(0,0,8)	3	2	4	2	3=3 (да) САК=8
21	8	J(0,3,12)	3	2	4	2	3=2 (нет)
22	9	J(2,3,13)	3	2	4	2	4=2 (нет)
23	10	S(3)	3	2	4	3	
24	11	J(0,0,8)	3	2	4	3	3=3 (да) САК=8
25	8	J(0,3,12)	3	2	4	3	3=3 (да) САК=12
26	12	T(2,0)	4	2	4	3	
27	13	Stop	4	2	4	3	

Глава 2. Программирование



«Программист обязан обладать способностью первоклассного математика к абстракции и логическому мышлению, эдисоновским талантом, уметь сооружать всё, что необходимо, из нуля и единицы. Он должен соединять в себе аккуратность банковского клерка с проницательностью разведчика, фантазию автора детективных романов с трезвой практичностью бизнесмена, кроме того, иметь вкус к коллективному труду…»

А.П. Ершов

В жизни, в профессиональной деятельности человек встречается с множеством практических задач. Для решения большинства из них можно использовать мощный и надёжный помощник – компьютер. Решение задач с использованием компьютера состоит из нескольких этапов:

1) постановка задачи;

2) анализ объекта моделирования и построение информационной модели;

3) алгоритмизация решения задачи – это выбор компьютерного исполнителя, выбор метода проектирования алгоритма, проектирование алгоритма. В этой работе мы предлагаем в качестве компьютерного исполнителя выбирать системы: Ершол, QBasic, Turbo Pascal, Turbo Delphi, Visual Basic.Net, Visual C#. Чаще всего алгоритм решения поставленной задачи на этом этапе описывается словесно или с использованием блок-схем;

4) создание компьютерной модели – реализация алгоритма на компьютере, кодирование алгоритма для выбранного исполнителя;

5) тестирование и отладка программы;

6) анализ результатов моделирования;

7) доработка программы для решения конкретных задач, составление документации для использования компьютерной модели.

В реализации этих этапов ведущая роль отводится программисту или группе программистов – для решения большой задачи. Программист – это связующее звено между компьютером и задачей, которую машине предстоит решать. У программиста очень ответственная роль: он должен представить любую задачу в виде программы.

Ведущими программистами мира разрабатываются правила создания больших программных комплексов с целью дисциплинировать процесс их разработки, снизить их сложность и стоимость.

Современными методологиями процедурного проектирования программ являются «Структурное программирование» и «Объектно-ориентированное программирование». Лучшими программистами 70-х годов (Дейкстра[9], Вирт[10], Хоар[11] и др.) были разработаны строгие правила проектирования программ, которые получили название – структурная методология. Дейкстра в 1969 году на международной конференции впервые использовал термин «структурное программирование». Метод *структурного программирования* основан на предположении, что код в модуле легче пишется, читается, если он сконструирован из фиксированного набора базовых структур организации действий, управляющих структур, с ограниченным использованием оператора GOTO. Доказано, что организовать действия в любой сложной программе можно используя линейную комбинацию основных управляющих структур: следования, условных и циклических

[9] Эдсгер Вибе Дейкстра (11.05.1930-6.08.2002) – выдающийся нидерландский ученый, идеи которого оказали огромное влияние на развитие технологий программирования.

[10] Никлаус Вирт (15.02.1934) – профессор Цюрихского технического университета, один из известных теоретиков в области разработки языков программирования (Паскаль, Модула-2, Оберон), лауреат премии Тьюринга 1984 года.

[11] Чарльз Энтони Ричард Хоар (11.01.1934) – профессор Оксфордского университета, специалист в области информатики и вычислительной техники. Наиболее известен как разработчик алгоритма «быстрой» сортировки.

структур. Другим принципом структурной методологии является принцип «разделяй и властвуй», принцип декомпозиции. Декомпозиция используется для разбиения программы на компоненты (модули), которые затем могут быть объединены, позволяя решить основную задачу. Модуль[12] – это конструкция, объединяющая и надежно скрывающая детали реализации определенной подзадачи, выполняемой модулем. Выполняемые локальные действия модулем должны быть гарантировано независимыми от влияния других частей основной программы при любых изменениях и коррекциях. *Основные концепции структурного программирования*: вся программа построена из модулей; модуль – это независимый блок, код которого физически и логически отделен от кода других модулей; каждый модуль имеет единственную точку входа/выхода; идентификаторы всех переменных и модулей должны быть смысловыми; идентификаторы, относящиеся к одному типу должны начинаться с одинакового префикса, указывающего этот тип; при кодировании использовать только стандартные управляющие конструкции; при оформлении программы использовать «ступенчатую» форму записи кода: вложенные операторы и серии команд, входящих в составные команды, при записи сдвигаются на несколько позиций влево относительно записи операторов, которым они принадлежат.

Визуальное объектно-ориентированное программирование является развитием технологии алгоритмического структурного программирования. *Объектно-ориентированное программирование* – это методология программирования, которая основана на представлении программы в виде совокупности объектов. Процесс разработки программы в среде визуального объектно-ориентированного программи-

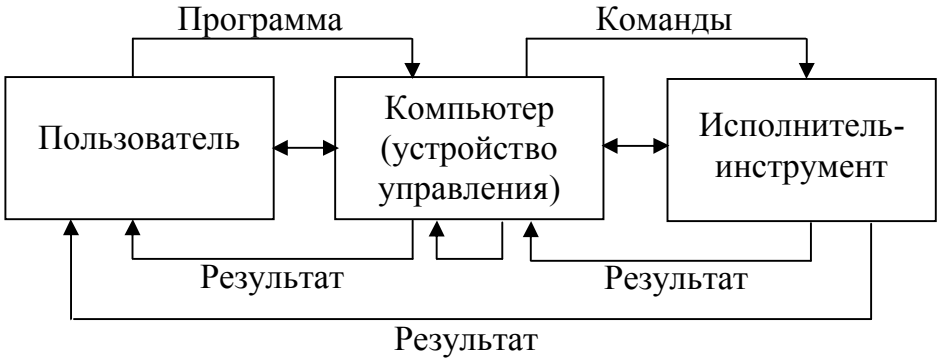
[12] Впервые специализированная синтаксическая конструкция *«модуль»* была предложена *Н. Виртом в 1975 г.* и включена в его новый язык Modula. После некоторой переработки этот новый язык был окончательно реализован в 1977 г. и получил название Modula-2. Впоследствии, аналогичные конструкции, с некоторыми отличиями, были включены и в другие языки программирования: Ada (1980 г.), Turbo Pascal.

рования сводится к выбору набора объектов и их свойств, заданию событий и процедур их обработки, которые в совокупности обеспечивают решение поставленной задачи.

На практике при создании крупных проектов рекомендуется применять обе методологии разработки программ: «Структурное программирование» и «Объектно-ориентированное программирование». Целесообразно сначала применять объектно-ориентированный подход для создания общей иерархии объектов, отражающих сущность программируемой задачи, а затем использовать структурную методологию для упрощения разработки программного кода.

Схема взаимодействия пользователя и исполнителя

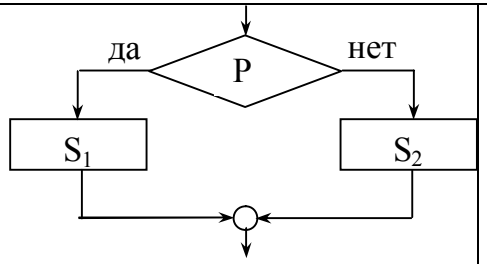
Как можно командовать исполнителем-инструментом? Почему составные команды организации действий в программах называются управляющими? Чтобы ответить на эти вопросы, учащиеся должны понимать, что программы составляются для компьютера, который без вмешательства человека может выполнять составленную программу, управляя исполнителем-инструментом (смотрите схему). Следовательно, в программе надо предусмотреть все возможные ситуации, возникающие при решении задачи, и запрограммировать какие команды компьютер должен отдать исполнителю-инструменту в возникающих ситуациях. Тогда при написании программ для компьютера и возникает необходимость в составных командах, *управляющих командах организации действий*: командах ветвления, командах цикла. Эти команды компьютер «научен» выполнять.

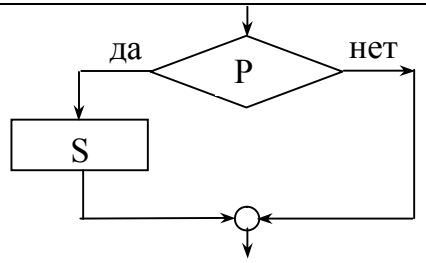


Пользователь передаёт компьютеру программу решения поставленной задачи. Компьютер выполняет программу, управляя исполнителем-инструментом. Часть времени компьютер расходует на обработку информации, не обращаясь к исполнителю. Пользователь получает результаты от компьютера и исполнителя – это, например, чертежи, картины, документы, расчёты.

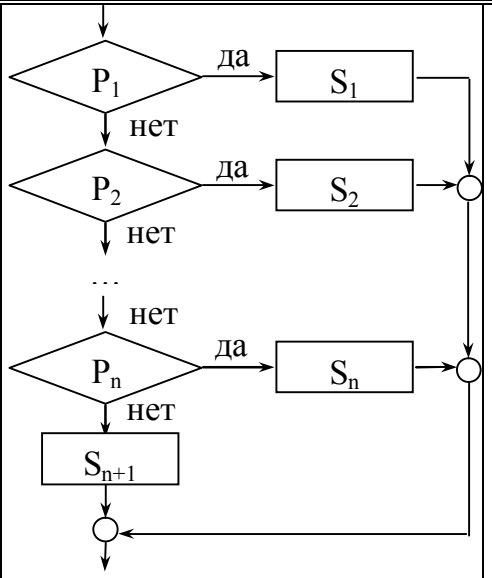
1. Кодирование управляющих команд организации действий на процедурных языках Ершол, QBasic, Turbo Pascal[13]

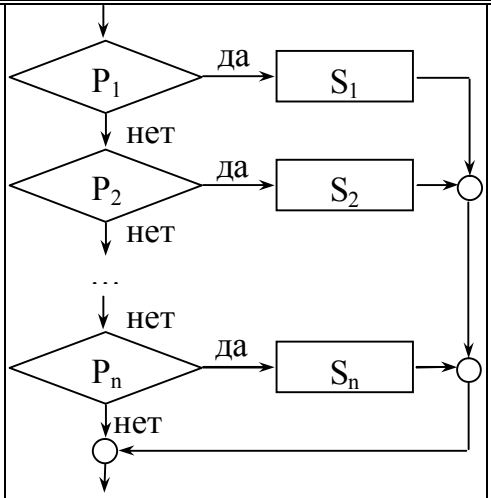
№ п/п	Название команды	Представление				
		Графическое	Turbo Pascal		QBasic	Ершол
			Серии S1,…,SN состоят из одной команды	Серии S1,…,SN содержат более одной команды		
Команды ветвления						

1	Команда ветвления в полной форме		If P then
			S_1
			else
			S_2;
				If P then
				begin
				S_1
				end
				else
				begin
				S_2
				end;
					If P then
					S_1
					else
					S_2
					End if
						если P то
						S_1
						иначе
						S_2
						все

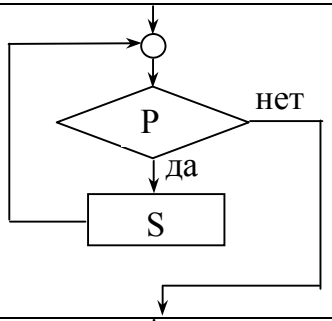
2	Команда ветвления в сокращенной (неполной) форме		If P then
			S_1;
				If P then
				begin
				S_1
				end;
					If P then
					S
					End if
						если P то
						S
						все

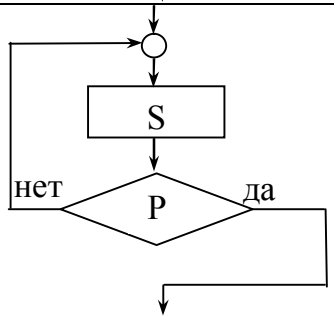
[13] В приведенных ниже шаблонах управляющих команд **P**, $\mathbf{P_1}$, … , $\mathbf{P_n}$ – проверяемые условия (логические выражения); **S**, $\mathbf{S_1}$, … , $\mathbf{S_n}$ – серии команд данного языка.

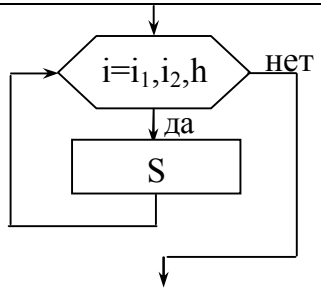
3	Команда выбора в полной форме		If P_1 then
			begin
			S_1;
			Goto K
			end;
			If P_2 then
			begin
			S_2;
			Goto K
			end;
			…
			If P_n then
			S_n
			else
			S_{n+1};
			K: <следующий за командой выбора оператор>
				If P_1 then
				begin
				S_1;
				Goto K
				end;
				If P_2 then
				begin
				S_2;
				Goto K
				end;
				…
				If P_n then
				begin
				S_n
				end
				else
				begin
				S_{n+1}
				end;
				K: <следующий за командой выбора оператор>
					If P_1 then
					S_1
					elseif P_2 then
					S_2
					elseif P_3 then
					S_3
					…
					elseif P_n then
					S_n
					else
					S_{n+1}
					End if
						выбор
						при P_1: S_1
						при P_2: S_2
						при P_3: S_3
						…
						при P_n: S_n
						иначе S_{n+1}
						все

4	Команда выбора в сокращенной форме		If P_1 then
			begin
			S_1;
			Goto K
			end;
			If P_2 then
			begin
			S_2;
			Goto K
			end;
			…
			If P_n then
			S_n;
			K: <следующий за командой выбора оператор>
				If P_1 then
				begin
				S_1;
				Goto K
				end;
				If P_2 then
				begin
				S_2;
				Goto K
				end;
				…
				If P_n then
				begin
				S_n;
				end;
				K: <следующий за командой выбора оператор>
					If P_1 then
					S_1
					elseif P_2 then
					S_2
					elseif P_3 then
					S_3
					…
					elseif P_n then
					S_n
					End if
						выбор
						при P_1: S_1
						при P_2: S_2
						при P_3: S_3
						…
						при P_n: S_n
						все

Циклические команды						

5	Команда повторения «пока» (цикл с предусловием)		While P do
			S;
				While P do
				begin
				S
				end;
					Do while P
					S
					Loop
						нц пока P
						S
						кц

6	Команда повторения «повторять S пока P не станет истинным» (цикл с постусловием)		Repeat S Until P;	Repeat S Until P;	Do S Loop until P	нц S кц при P

7	Команда повторения «для» (цикл с параметром)		For i:=i_1 to i_2 do
			S;
			For i:=i_1 downto i_2 do
			S;
				For i:=i_1 to i_2 do
				begin
				S
				end;
				For i:=i_1 downto i_2 do
				begin
				S
				end;
					For i=$i_{нач}$ to $i_{кон}$ [step шаг]
					S
					Next i
						нц для i от $i_{нач}$ до $i_{кон}$ [шаг $i_{шаг}$]
						S
						кц

Оператор варианта на языках QBasic и Turbo Pascal

QBasic	Turbo Pascal

Select Case <выражение>
Case <значение1>
<серия операторов, которые выполняются, когда значение выражения равно значению1>
Case <значение2>
<серия операторов, которые выполняются, когда значение выражения равно значению2>
…
Case <значение_n>
<серия операторов, которые выполняются, когда значение выражения равно значению_n>
[Case Else
<серия операторов, которые выполняются, когда значение выражения не равно значению1, значению2, … , значению_n>
End Select
	Case <выражение> of
	<список1, значений выражения, перечисленных через запятую>:
	begin
	S_1
	end;
	<список2, значений выражения, перечисленных через запятую>:
	begin
	S_2
	end;
	…
	<список_n, значений выражения, перечисленных через запятую>:
	begin
	S_n
	end
	[else
	begin S_{n+1} end;]
	end;

2. Коды программ решения задач 10-36 на языках Ершол, QBasic, Turbo Pascal[14]

Задача 10, стр. 33 (Ершол):

```
алг БИД (арг вещ a, b, рез вещ x)
 нач  вещ R
   |ввод операндов a и b организует сама
   |система Ершол, они описаны как арг
```

Задача 10, стр. 33 (Turbo Pascal):

```
Program BID;{Заголовок}
Uses CRT; {Подсоединение модуля CRT}
{Блок описания переменных}
```

[14] В программах информация, записанная после символов «'» в языке QBasic, «|» – в Ершоле, между символами «{», «}» – в Turbo Pascal'е является комментарием и при наборе программы её лучше опустить. В целях компактной записи заголовки программ записаны в две строки, что запрещено. При использовании различных систем «Кумир» возможно возникновение ошибки в заголовке алгоритма. В этом случае необходимо описать данные после служебного слова «нач» и реализовать ввод данных с помощью операторов «вывод», «ввод». В некоторых версиях QBasic на одной строке писать несколько операторов, разделенных «:», нельзя. Тогда необходимо написать каждый оператор в одной строке, опустив знак «:».

```
  R:=a-b      |найдем разность чисел a и b
  |команда ветвления в полной форме
  если R<0    |если условие R<0 истинно,
   то x:=b    |то x присваивается
              |значение b,
   иначе x:=a| иначе x принимает зн. a
    все
  |вывод операнда x организует сама
  |система Ершол, он описан как рез
кон
```

Задача 10, стр. 33 (QBasic)

```
REM Большее из двух
'Блок описания переменных a, b, R, x
DIM a AS SINGLE, b AS SINGLE
DIM R AS SINGLE, x AS SINGLE
CLS  'Очистка экрана
'Ввод чисел a и b
INPUT "Введите два числа a и b: "; a, b
'Найдена разность чисел a и b
R = a – b
'Сравниваем разность с 0
IF R < 0 THEN
'Если разность меньше 0, то большее
'число b
                x = b
          ELSE       'иначе,
'если разность больше или равна 0,
'то большее число a
                x = a
END IF
'Блок вывода результата
PRINT "Большее из чисел ";
PRINT USING "###.###"; a;:PRINT " и";
PRINT USING "###.###"; b;:PRINT " ---";
PRINT USING "###.###"; x
END                          'Конец
```

Задача 11, стр. 34 (Turbo Pascal)

```
Program Kvadr_ur;
Uses CRT;
Var a,b,c,d,r,x1,x2:real;
Begin
   Write('Введите коэффициенты a,b,c:');
   Readln(a,b,c);
Var
  a,b,R,x:real; {a, b, R, x – веществен-
ные}
Begin
  ClrScr; {Очистка экрана}
  {Вывод на экран подсказки}
  Write('Введите два числа: ');
  Readln(a,b); {Ввод чисел a и b}
  R:=a-b; {Нахождение разности чисел
           a и b}
  {Выбор большего из чисел a и b}
  If R<0 then x:=b else x:=a;
  {Блок печати}
  Writeln('max (',a:6:3,' и ',b:6:3,' - ) ',
x:6:3)
End.
```

Задача 11, стр. 34 (Ершол)

```
алг КВУР (арг вещ a,b,c)
 нач вещ R,D,x1,x2 |данные R, D -
  |промежуточные величины
  |ввод аргументов организует
  |система Ершол
  D:=b*b-4*a*c|оператор
              |присваивания
  | команда ветвления в полной форме
  если D>0
   то
    |если D>0, то вычисляем корни
    R:=sqrt(D)  |квадратного уравнения
     x1:=(-b-R)/(2*a); x2:=(-b+R)/(2*a)
     |выводим знач. x1 и x2
    вывод нс, "x1= ",x1,нс,"x2= ",x2
    иначе
    |выводим сообщение
     вывод нс
     вывод "корни не вычисляем
     (D<=0)"
  все
кон
```

Задача 11, стр. 34 (QBasic)

```
REM Квадратное уравнение
DIM a AS SINGLE, b AS SINGLE
DIM c AS SINGLE, d AS SINGLE
```

```
    d:=b*b-4*a*c;
    If d>0 then
      begin
          r:=sqrt(d);
          x1:=(-b-r)/(2*a);
          x2:=(-b+r)/(2*a);
          Writeln('x1=',x1,' x2= ',x2)
      end
            else
       Writeln('Корни не вычисляем
(D<=0)')
End.
```

Задача 12, стр. 34 (Ершол)

```
алг площадь треугольника (рез вещ s)
 дано
 надо
нач вещ a,b,c, вещ p,k
  |данные p и k-промежуточные
  |величины
  |оператор вывода сообщения
  вывод "введите знач. a"
  ввод a       |оператор ввода операнда a
  |оператор вывода сообщения
 вывод нс,"введите знач. b"
  ввод b       |оператор ввода операнда b
 вывод нс,"c= " |оператор вывода
  |сообщения
  ввод c       |оператор ввода операнда c
  p:=(a+b+c)/2  |оператор присваивания
 |оператор присваивания
 k:=p*(p-a)*(p-b)*(p-c)
  если k>0
    то
      s:=sqrt(k)
    иначе
      s:=-1
  все
  |вывод значения s организует система
  |Ершол
кон
```

Задача 13, стр. 35 (Ершол)

```
алг БИТ (арг вещ a,b,c,d, рез вещ max)
 дано
 надо
```

```
DIM r AS SINGLE, x1 AS SINGLE
DIM x2 AS SINGLE
INPUT "Введите коэфф. a,b,c: "; a,b,c
d = b * b - 4 * a * c
IF d > 0 THEN
      r = SQR(d)
      x1 = (-b - r) / (2 * a)
      x2 = (-b + r) / (2 * a)
      PRINT "x1="; x1, " x2="; x2
            ELSE
      PRINT "Корни не вычисляем
(D<=0)"
END IF
END
```

Задача 12, стр. 34 (Turbo Pascal)

```
Program treug;
Uses CRT;
Var a,b,c,p,k,s:real;
Begin
    Write('Введите a,b,c: ');
    Readln(a,b,c);
    p:=(a+b+c)/2;
    k:=p*(p-a)*(p-b)*(p-c);
    If k>0 then
            s:=sqrt(k)
         else
            s:=-1;
    Writeln(s)
End.
```

Задача 12, стр. 34 (QBasic)

```
REM Площадь треугольника
DIM a AS SINGLE, b AS SINGLE
DIM c AS SINGLE, p AS SINGLE
DIM k AS SINGLE, s AS SINGLE
INPUT "Введите a,b,c: "; a, b, c
p = (a + b + c) / 2
k = p * (p - a) * (p - b) * (p - c)
IF k > 0 THEN
          s = SQR(k)
       ELSE
          s = -1
END IF
PRINT s
END
```

```
нач
 |ввод данных a,b,c организует система
 |Ершол
 |команда ветвления в полной форме
 если a<b     |проверяется условие a<b,
  то          |если условие истинно, то
    max:=b    |max присваивается
              |значение b,
  иначе       |если условие ложно, то
    max:=a    |max присваивается
              |значение a
 все
 |команда ветвления в сокращенной
 |форме
 если max<c |проверяется условие
            |max<c,
  то        |если условие истинно, то
  max:=c    |max присваивается
            |значение c,
  |иначе команда ветвления никаких
  |действий не предписывает
 все
 |вывод результата max организует
 |система Ершол
кон
```

Задача 13, стр. 35 (QBasic)

```
REM Большее из трех
DIM a AS SINGLE, b AS SINGLE
DIM c AS SINGLE, max AS SINGLE
INPUT "Введите a,b,c: "; a, b, c
IF a < b THEN
        max = b
    ELSE
        max = a
END IF
IF max < c THEN max = c
PRINT max
END
```

Задача 14, стр. 35 (Turbo Pascal способ 1)

```
Program Func;
Uses CRT;
Var x,y:real;
```

Задача 13, стр. 35 (Turbo Pascal)

```
Program BIT;
Uses CRT;
Var a,b,c,max:real;
Begin
   Write('Введите a,b,c: ');
   Readln(a,b,c);
   If a<b then
         max:=b
      else
         max:=a;
   If max<c then
         max:=c;
   Writeln(max)
End.
```

Задача 14, стр. 35 (Ершол способ 1)

```
алг функция (арг вещ x, рез вещ y)
 дано
 надо
нач
 |ввод знач. операнда x организует
 |система Ершол
 |вложенное ветвление
 |команда ветвления в полной форме
 если x>22 |проверяется условие
           |x>22,
  то |если оно истинно, то y присваи-
     |вается значение 1/(x*x+1),
   y:=1/(x*x+1)
  иначе       |иначе проверяется
  если x<=3   |условие x<=3, если оно
    то        |истинно, то y присваивается
     y:=x-1   |значение x-1,
    иначе     |иначе 3<x<=22,
    |y присваивется значение x**3+2
     y:=x**3+2
  все
 все
 |вывод значения y
 |организует система Ершол
кон
```

Задача 14, стр. 35 (QBasic способ 1)

```
Begin
    Write('Введите x: ');
    Readln(x);
    If x>22 then
                y:=1/(x*x+1)
             else
                If x<=3 then
                            y:=x-1
                         else
                            y:=x*x*x+2;
    Writeln('x=',x,' y=',y)
End.
```

Задача 14, стр. 35 (Ершол способ 2)

```
алг функция (арг вещ x, рез вещ y)
 дано
 надо
нач
  |ввод знач. операнда x организует
  |система Ершол
  |команда выбора в полной форме
  выбор
    |x<=3-истинно, то y присваивается
    |значение x-1
    при x<=3: y:=x-1
    | 3<x<=22-истинно, то y принимает
    |значение x**3+2,
    при x<=22: y:=x**3+2
    | иначе x>22, y принимает значение
    |1/(x*x+1)
    иначе y:=1/(x*x+1)
  все
  |вывод результата y организует
  |система Ершол
кон
```

Задача 14, стр. 35 (QBasic способ 2)

```
REM Значение функции
DIM y AS SINGLE, x AS SINGLE
INPUT "Введите x: "; x
IF x <= 3 THEN
                y = x - 1
       ELSEIF x <= 22 THEN
                            y = x * x * x + 2
                         ELSE
                            y = 1 / (x * x + 1)
```

```
REM Значение функции
DIM x AS SINGLE, y AS SINGLE
INPUT "Введите x: "; x
IF x > 22 THEN
           y = 1 / (x * x + 1)
       ELSE   IF x <= 3 THEN
                    y = x - 1
                  ELSE
                    y = x * x * x + 2
                END IF
END IF
PRINT "x="; x; " y="; y
END
```

Задача 14, стр. 35 (Turbo Pascal способ 2)

```
Program Func;
Uses CRT;
Label k;
Var y:real; x: integer;
Begin
    Write('Введите x: ');
    Readln(x);
    If x<=3 then begin
                    y:=x-1; goto k
                 end;
    If x<=22 then begin
                    y:=x*x*x+2; goto k
                  end;
     y:=1/(x*x+1);
 k:   Writeln('x=',x,' y=',y)
End.
```

Задача 15, стр. 37 (Turbo Pascal)

```
Program Segment;
Uses CRT;
Var
  a,b,c,d,x,y:real;
  r:0..1;
Begin
    Write('Введите a,b,c,d: ');
    Readln(a,b,c,d);
    If a>=c then
               x:=a
            else
```

```
END IF
PRINT "x="; x; " y="; y
END
```

Задача 15, стр. 37 (Ершол)

```
алг пересечение отрезков (арг вещ
                          a,b,c,d, рез вещ R)
нач вещ x,y |промежуточные данные
            |x и y
 | |ввод  данных a,b,c,d организует
 | |система Ершол
 | |первая команда ветвления в полной
 | |форме
 | если a>=c  |если условие a>=c истинно,
 |  | то x:=a   |то x присваивается
 |  |           |значение a,
 |  | иначе x:=c |иначе значение c
 |  все
 | |вторая команда ветвления в полной
 | |форме
 |  если b>=d |если условие b>=d
 |            |истинно,
 |  | то  y:=d |то y присваивается значение
 |            |d,
 |  | иначе y:=b|иначе – значение b
 |  все
 | |третья команда ветвления в полной
 | |форме
 |  если  x<=y |если условие x<=y
 |             |истинно,
 |  | то R:=1   |то R присваивается
 |             |значение 1,
 |  |   вывод нс, "[",x,";",y,"]"
 |  | иначе R:=0| иначе – значение 0
 |  все
 | |вывод операнда R организует система
 | |Ершол
кон
```

Задача 16, стр. 37 (Ершол)

```
алг сумма кубов (рез цел s)
 дано
 надо
нач цел n
 | |задание начальных значений
```

```
                x:=c;
    If b>=d then
                y:=d
             else
                y:=b;
    If x<=y then
            begin
               r:=1; Writeln(r,x,y)
            end
         else
            begin
               r:=0; Writeln(r)
            end
End.
```

Задача 15, стр. 37 (QBasic)

```
REM Пересечение отрезков
DIM a AS SINGLE, b AS SINGLE
DIM c AS SINGLE, d AS SINGLE
DIM x AS SINGLE, y AS SINGLE
DIM r AS INTEGER
INPUT "Введите a,b,c,d: "; a, b, c, d
IF a >= c THEN
           x = a
        ELSE
           x = c
END IF
IF b >= d THEN
           y = d
        ELSE
           y = b
END IF
IF x <= y THEN
           r = 1
           PRINT r, x, y
        ELSE
           r = 0
           PRINT r
END IF
END
```

Задача 17, стр. 39 (Ершол)

```
алг корень третьей степени (арг вещ x,
                            арг вещ e, рез вещ yn)
нач  вещ дельта,вещ yn1
 | |ввод аргументов (x, e) и вывод
```

```
 |переменным s и n
 s:=0     |s - текущая сумма
 n:=1     |n - номер текущего слагаемого
|цикл с предусловием,
нц пока n<=100       |проверка условия,
  | s:=s+n*n*n |пока условие истинно
  | n:=n+1     |выполняются команды
               |тела цикла
 кц
 |вывод значения s организует система
 |Ершол, т.к. операнд s описан как рез
кон
```

Задача 16, стр. 37 (Turbo Pascal)

```
Program Summa;
Uses CRT;
Var s,n: integer;
Begin
    s:=0; n:=1;
    Repeat
        s:=s+n*n*n;
        n:=n+1
    Until n>100;
    Writeln('s=',s)
End.
```

Задача 16, стр. 37 (QBasic)

```
REM Сумма кубов
DIM s AS LONG, n AS INTEGER
s = 0 : n = 1
DO
  s = s + n * n * n : n = n + 1
  PRINT s, n
LOOP UNTIL n > 100
PRINT "s="; s
END
```

Задача 17, стр. 39 (Turbo Pascal)

```
Program Root;
Uses CRT;
Var x,e,y1,y2,d:real;
Begin
    Write('Введите x,e');
    Readln(x,e);
    y1:=x;
    Repeat
```

```
  |результата (yn) организует система
  |Ершол, так как эти операнды
  |описаны в заголовке как арг и рез
  |задание начальных значений
  |yn, yn1, дельта
  yn:=x; yn1:=2/3*(yn+x/(2*yn*yn))
  дельта:=abs(yn-yn1)
  |цикл с предусловием
  |проверка условия окончания цикла,
  |пока условие истинно
 нц пока дельта>e      | yn получает
   | yn:=yn1           |значение yn1,
   |вычисляется новое значение yn1,
   | yn1:=2/3*(yn+x/(2*yn*yn))
   |вычисляется погрешность
   | дельта:=abs(yn-yn1)
 кц
кон
```

Задача 18, стр. 40 (Ершол)

```
алг Фибоначчи
 дано
 надо
нач  цел n,цел c, цел a,цел b, цел k
  |блок ввода
  вывод "введите номер иск. чл. Ф."
  ввод n
  |первому числу Фибоначчи – a –
  |присвоим значение 1, второму –
  |числу b – присвоим значение 1
  a:=1;b:=1
  |параметру цикла k присвоим зн. 3 -
  |номер первого искомого числа Фиб.
  k:=3
  |цикл с предусловием
  нц пока k<=n |проверка условия,
   | c:=a+b  |пока условие истинно
   | a:=b    |находят c - следующий чл.
   |Фиб. и переприсваивают значения
   | b:=c    | предыдущих a и b
   |находят номер следующего
   |искомого числа Фибоначчи
   | k:=k+1
  кц
  |блок вывода
```

```
        y2:=2/3*(y1+x/(2*y1*y1));
        d:=abs(y2-y1);
        y1:=y2
    Until d<=e;
    Writeln(y1:7:5)
End.
```

Задача 17, стр. 39 (QBasic)

```
REM Корень третьей степени
DIM x AS SINGLE, e AS SINGLE
DIM y1 AS SINGLE, y2 AS SINGLE
DIM d AS SINGLE
INPUT "Введите x, e"; x, e
y1 = x
DO
    y2 = 2 / 3 * (y1 + x / (2 * y1 * y1))
    d = ABS(y2 - y1) :   y1 = y2
LOOP UNTIL d <= e
PRINTUSING "###.##"; y1
END
```

Задача 18, стр. 40 (Turbo Pascal)

```
Program Fibonacci;
Uses CRT;
Var a,b,c,k,n:integer;
Begin
    Write('Введите n ');
    Readln(n);
    a:=1; b:=1; k:=3;
    Repeat
        c:=a+b; a:=b; b:=c;
        k:=k+1
    Until k>n;
    Writeln(c)
End.
```

Задача 18, стр. 40 (QBasic)

```
REM Число Фибоначчи
DIM a AS INTEGER, b AS INTEGER
DIM c AS INTEGER, k AS INTEGER
DIM n AS INTEGER
INPUT "Введите n "; n
a = 1
b = 1
k = 3
DO
```

```
| вывод нс,"число Фибоначчи равно",c
кон
```

Задача 19, стр. 41 (Ершол)

```
алг косинус_x
 нач вещ x,e,s,a,R, цел n
 | | блок ввода
 | вывод "введите значение x -"
 | ввод x
 | вывод нс,  "введите значение e -"
 | ввод e
 |задание начальных a,s,x,R,n
 | a:=1; s:=1;x:=x*arctg(1)*4/180;
 | R:=-x*x; n:=1
 | |цикл с предусловием
 | нц пока abs(a)>e   |проверка условия,
 | | |пока условие истинно
 | | |выполняются команды тела цикла:
 | | |вычисляется новый член ряда,
 | | a:=a*R/((2*n-1)*2*n)
 | | s:=s+a            |новая сумма ряда,
 | | n:=n+1            |увеличивается n
 | кц
 | |блок вывода
 | вывод нс, "x= ",x
 | вывод нс,"вычисленное значение"
 | вывод "cos(x)=", s
 | вывод нс,"зн. встроенной функции"
 | вывод "cos(x)=",cos(x)
кон
```

Задача 19, стр. 41 (QBasic)

```
REM Косинус угла
DIM x AS SINGLE, e AS SINGLE
DIM a AS SINGLE, s AS SINGLE
DIM r AS SINGLE, n AS INTEGER
INPUT "Введите x, e "; x, e
a = 1 : s = 1 : x = x * ATN(1) * 4 / 180
r = -1 * x * x : n = 1
DO
  a = a * r / ((2 * n - 1) * 2 * n)
  s = s + a
  n = n + 1
LOOP UNTIL ABS(a) <= e
PRINT x, s
END
```

```
  c = a + b : a = b : b = c
  k = k + 1
LOOP UNTIL k > n
PRINT c
END
```

Задача 19, стр. 41 (Turbo Pascal)

```
Program Cosinus;
Uses CRT;
Var  x,e,a,s,r:real;
    n:integer;
Begin
    Write('Введите x,e ');
    Readln(x, e);
    a:=1;
    s:=1;
    x:=x*pi/180;
    r:=-1*x*x;
    n:=1;
    Repeat
        a:=a*r/((2*n-1)*2*n);
        s:=s+a;
        n:=n+1
    Until abs(a)<=e;
    Writeln(x,s)
End.
```

Задача 20, стр. 41 (Ершол)

```
|заголовок
алг полет мухи(арг вещ e,d,vm,v1,v2)
 дано
 надо
нач лит l, вещ y,t,x,s
  |ввод операндов (e, d, vm, v1, v2)
  |организует система Ершол, они
  |описаны в заголовке как аргументы
  y:=d; s:=0|задание начальных значений
  l:="к поезду из B" |операндов y, s, l
  |цикл с предусловием,
  |пока y>e истинное высказывание
  нц пока y>e |выполнются команды
              | тела цикла,
    |если муха летит к поезду B,
    если l="к поезду из B"
      то          |то далее она полетит
      l:="к поезду из A" |к поезду A,
```

Задача 20, стр. 41 (Turbo Pascal)

```
Program Fly;
Uses CRT;
Var  d,v1,v2,vm,y,s,e,t,x:real;
    l:string;
Begin
    Write('Введите d,v1,v2,vm,e ');
    Readln(d,v1,v2,vm);
    y:=d;
    s:=0;
    l:='К поезду изB';
    While y>e do
     begin
       If l='К поезду из B' then
                        begin
                           l:='К поезду из A';
                            t:=y/(v2+vm)
                        end
                       else
                        begin
                           l:='К поезду из B';
                            t:=y/(v1+vm)
                        end;
       x:=t*vm; s:=s+x;
       y:=y-t*(v1+v2);
       Writeln(t,x,s)
     end
End.
```

Задача 20, стр. 41 (QBasic)

```
REM Сверхбыстрая муха
DIM d AS SINGLE, v1 AS SINGLE
DIM v2 AS SINGLE, vm AS SINGLE
DIM y AS SINGLE, s AS SINGLE
DIM e AS SINGLE, t AS SINGLE
DIM x AS SINGLE, l AS STRING
INPUT "Вв. d,v1,v2,vm,e";d,v1,v2,vm,e
y = d
s = 0
l = "К поезду из B"
WHILE y > e
   IF l = "К поезду из B" THEN
                        l = "К поезду из A"
                        t = y / (v2 + vm)
                    ELSE
```

```
        |определим время полета мухи до
        |встречи с поездом В,
         t:=y/(vm+v2)
        иначе |иначе она летит к поезду А,
        |далее она полетит к поезду В,
         l:="к поезду из В"
        |определим время полета мухи до
        |встречи с поездом А
         t:=y/(vm+v1)
      все
      |блок вывода операндов t, x, s, y
      |(можно выводить только x и s)
      вывод нс,"t=",t
       x:=t*vm; вывод "x=",x
       s:=s+x; вывод "s=",s
       y:=y-t*(v1+v2); вывод "y=",y
  кц
кон
```

Задача 21, стр. 46 (Ершол)

```
алг произведение1 (арг цел n,
                   арг вещ таб a[1:n], рез вещ П)
нач цел k
  |ввод количества элементов n и
  |элементов массива a организует
  |система Ершол
  |задание начальных значений данных
  П:=1;k:=1                          |П и k
  |цикл с предусловием
  нц пока k<=n |проверяется условие
               |k<=n,
    |пока условие истинно вычисляется
    |произведение k элементов массива,
     П:=П*a[k]
    |увеличивается номер (k)
     k:=k+1        |очередного элемента
  кц
  |если k<=n ложно, т.е. k>n, то
  |выполнится вывод произведения П,
  |вывод произведения организует
  |система Ершол
кон
```

Задача 21, стр. 46 (Turbo Pascal)

```
Program Array_element_product;
Uses CRT; Const n=10;
                    l = "К поезду из В"
                    t = y / (v1 + vm)
  END IF
  x = t * vm
  s = s + x
  y = y - t * (v1 + v2)
  PRINT t, x, s
WEND
END
```

Задача 21, стр. 46 (QBasic)

```
REM Произведение элементов массива
DIM n AS INTEGER, k AS INTEGER
DIM p AS SINGLE
CLS
INPUT "Введите количество чисел n ";
n
DIM a(1 TO n) AS SINGLE
FOR k = 1 TO n
 PRINT "Введите a("; k; ") ";
 INPUT a(k)
NEXT k
p = 1
k = 1
WHILE k <= n
 p = p * a(k)
 k = k + 1
WEND
PRINT "p="; p
END
```

Задача 22, стр. 46 (Ершол)

```
алг произведение2 (арг цел n,m)
 нач вещ таб a[1:m,1:n], вещ таб b[1:n],
     цел i,j,вещ таб c[1:m]
  |ввод значений данных n и m
  |организует система Ершол
  |блок ввода элементов массива a
  нц для i от 1 до m шаг 1
    нц для j от 1 до n шаг 1
      вывод "a[",i,",",j,"]="
      ввод a[i,j]
    кц
    |перевод курсора на новую строку
    вывод нс
  кц
```

```
Var
  k:integer;
  p:real;
  a:array [1..n] of real;
Begin
  ClrScr;
  Write('Введите количество чисел n ');
  Readln(n);
  For k:=1 to n do
    begin
      Write('Введите a[',k,'] ');
      Readln(a[k])
    end;
  p:=1;
  k:=1;
  While k<=n do
    begin
      p:=p*a[k]; k:=k+1
    end;
  Writeln('p=',p)
End.
```

Задача 22, стр. 46 (QBasic)

```
REM Произведение матриц
DIM i AS INTEGER, j AS INTEGER
DIM n AS INTEGER, m AS INTEGER
INPUT "Введите m, n "; m, n
DIM c(1 TO m) AS SINGLE
DIM b(1 TO n) AS SINGLE
DIM a(1 TO m, 1 TO n) AS SINGLE
CLS
FOR i = 1 TO m
  FOR j = 1 TO n
    PRINT "Введите a("; i; ","; j; ") =";
    INPUT a(i, j)
  NEXT j
NEXT i
FOR i = 1 TO n
  PRINT "Введите b("; i; ")= ";
  INPUT b(i)
NEXT i
i = 1
DO
  c(i) = 0 :   j = 1
  DO
```

```
|блок ввода элементов массива b
нц для i от 1 до n шаг 1
| вывод "b[",i,"]="
| ввод b[i]
кц
|блок вывода элементов массива a
вывод нс, "массив a",нс
нц для i от 1 до m шаг 1
| нц для j от 1 до n шаг 1
| | вывод a[i,j]
| кц
|перевод курсора на новую строку
| вывод нс
кц
|блок вывода элементов массива b
вывод  "массив b"
нц для i от 1 до m шаг 1
| вывод нс, b[i]
кц
нц для i от 1 до m шаг 1
| c[i]:=0                  |блок получения
| нц для j от 1 до n шаг 1 |элементов
| | c[i]:=c[i]+a[i,j]*b[j]   |массива c
| кц
кц
|блок вывода элементов массива c
вывод нс, "массив c"
нц для i от 1 до m шаг 1
| вывод нс, c[i]
кц
кон
```

Задача 22, стр. 46 (Turbo Pascal)

```
Program Matrix_product;
Uses CRT;
Const n1=10; m1=10;
Var
  i,j,n,m:integer;
  c:array [1..m1] of real;
  b:array [1..n1] of real;
  a:array[1..m1,1..n1] of real;
Begin
  ClrScr;
  Write('Введите m, n ');
  Readln(m,n);
```

```
    c(i) = c(i) + a(i, j) * b(j) :  j = j + 1
  LOOP UNTIL j > n
  i = i + 1
LOOP UNTIL i > m
FOR i = 1 TO m
  PRINT "c("; i; ")="; c(i)
NEXT i
END
```

Задача 23, стр. 48 (Ершол)

```
алг положительные элементы (арг цел n,
                            арг вещ таб a[1:n])
 нач цел i, цел k
 | |ввод аргумента n и элементов
 | |массива a организует система Ершол
 | |задание начальных значений
 | |данных i,  k
 | k:=0; i:=1
 | |команда повторения с предусловием
 | |проверяется условие i<=n и k=0, пока
 | |условие истинно
 | нц пока i<=n и k=0 | выполняется
 | | |команда ветвления в неполной
 | | |форме:
 | | если a[i]>0 |проверяется условие
 | |                 |a[i]>0,
 | | | то       |если оно истинно, то  k
 | | |  k:=1  |присваивается значение 1
 | |  все
 | | |i увеличивается на единицу, это
 | | |индекс просматриваемого элемента
 | | |массива
 | |  i:=i+1
 | кц
 | вывод нс,"k=",k |вывод значения k
кон
```

Задача 23, стр. 48 (Turbo Pascal)

```
Program Positive_number;
Uses CRT;
Const n1=10;
Var
  i,n,K:integer;
  a:array [1..n1] of real;
Begin
  ClrScr;
  For i:=1 to m do
    For j:=1 to n do
      begin
        Write('Введите a[',i,',',j,'] ');
        Readln(a[i,j])
      end;
  For i:=1 to n do
    begin
      Write('Введите b[',i,'] ');
      Readln(b[i])
    end;
  i:=1;
  Repeat
    c[i]:=0; j:=1;
    Repeat
      c[i]:=c[i]+a[i,j]*b[j]; j:=j+1
    Until j>n;
    i:=i+1;
  Until i>m;
  For i:=1 to m do
    Writeln('c[',i,']=',c[i])
End.
```

Задача 23, стр. 48 (QBasic)

```
REM Положительные числа
DIM i AS INTEGER, n AS INTEGER
DIM K AS INTEGER
CLS
INPUT "Введите n "; n
DIM a(1 TO n) AS SINGLE
FOR i = 1 TO n
  PRINT "Введите a("; i; ") ";
  INPUT a(i)
NEXT i
K = 0
i = 1
DO
  IF a(i) > 0 THEN K = 1
  i = i + 1
LOOP UNTIL (i > n) OR (K = 1)
PRINT "K="; K
END
```

Задача 24, стр. 49 (Turbo Pascal)

```
Program Searching_the_letter;
Uses CRT;
```

```
  Write('Введите n ');
  Readln(n);
  For i:=1 to n do
    begin
      Write('Введите a[',i,'] ');
      Readln(a[i])
    end;
  K:=0; i:=1;
  Repeat
    If a[i]>0 then K:=1;
    i:=i+1
  Until (i>n) or (K=1);
  Writeln('K=',K)
End.
```

Задача 24, стр. 49 (Ершол)

```
алг поиск буквы (арг цел n,
                 арг лит таб w[1:n], арг сим v)
 дано
 надо
нач цел i, цел k
 |ввод операнда n и элементов массива
 |w организует Ершол
 |задание начальных значений
 k:=0 ;i:=1|операндов k и i
 |запишем команду повторения с
 |предусловием
 |проверяем условие i<=n и k=0,
 нц пока i<=n и k=0 |пока оно истинно
  | если v=w[i] |выполняется команда
  |          |ветвления в полной форме:
  |  | то  |проверим условие v=w[i],
  |  |если оно истинно, то k присвоим
  |  |  k:=i    |значение 1 иначе i
  |  | иначе  i:=i+1 |увеличим на 1
  | все
 кц
 вывод нс, "k=",k
кон
```

Задача 24, стр. 49 (QBasic)

```
REM Поиск буквы
CONST n = 5
DIM i AS INTEGER, K AS INTEGER
DIM w  AS STRING *n, v AS STRING *1
CLS
Const n=5;
Var
  i,K:integer;
  w:string[n];
  v:char;
Begin
  ClrScr;
  Write('Введите слово ');
  Readln(w);
  Write('Введите букву ');
  Readln(v);
  K:=0;
  i:=1;
  Repeat
    If v=w[i] then K:=i else i:=i+1
  Until (i>n) or (K<>1);
  Writeln('K=',K)
End.
```

Задача 25, стр. 49 (Ершол)

```
алг количество положительных (арг
                 цел n,K,арг вещ таб a[1:n])
 нач  цел i,l,m
 |  |ввод данных n, K и элементов
 |  |массива организует система Ершол
 |  l:=0; i:=1  |задание нач. значения l и i
 |  | команда повторения "пока" с
 |  |проверкой условия i<=n и l<K,
 |  нц пока i<=n и l<K     |пока условие
 |  |             |истинно выполняются
 |  |  если a[i]>0            |команды тела
 |  |  | то     |цикла: команда ветвления и
 |  |  |  l:=l+1
 |  |  все
 |  |  i:=i+1      |команда присваивания
 |  кц
 |  если l=K
 |  | то m:=i-1
 |  | иначе m:=0
 |  все
 |  вывод нс,"m=",m
кон
```

Задача 25, стр. 49 (QBasic)

```
REM Количество положительных
REM элементов
```

```
INPUT "Введите слово "; w
INPUT "Введите букву "; v
K = 0
i = 1
DO
  IF v = MID$(w, i, 1) THEN
                              K = i
                           ELSE
                              i = i + 1
  END IF
LOOP UNTIL (i > n) OR (K <> 0)
PRINT "K="; K
END
```

Задача 25, стр. 49 (Turbo Pascal)

```
Program Amount_positive_element;
Uses CRT;
Const n1=10;
Var
  n,i,K,l,m:integer;
  a:array[1..n1] of real;
Begin
  ClrScr;
  Write('Введите n '); Readln(n);
  Write('Введите K '); Readln(K);
  For i:=1 to n do
    begin
      Write('Введите a[',i,'] '); Readln(a[i])
    end;
  l:=0; i:=1;
  Repeat
    If a[i]>0 then l:=l+1;
    i:=i+1
  Until (i>n) or (l=K);
  If l=K then m:=i-1 else m:=0;
  Writeln('m=',m)
End.
```

Задача 26, стр. 49 (Ершол)

```
алг сдвиг в массиве (арг цел n,
                     арг рез вещ таб a[1:n])
нач вещ c, цел i
  |ввод операнда n и элементов массива
  |a организует система Ершол
  |значение a[n] сохраняется
  |в переменной c
DIM n AS INTEGER, i AS INTEGER
DIM K AS INTEGER, l AS INTEGER
DIM m AS INTEGER
CLS
INPUT "Введите n "; n
DIM a(1 TO n) AS SINGLE
INPUT "Введите K "; K
FOR i = 1 TO n
  PRINT "Введите a("; i; ") ";
  INPUT a(i)
NEXT i
l = 0
i = 1
DO
  IF a(i) > 0 THEN l = l + 1
  i = i + 1
LOOP UNTIL (i > n) OR (l = K)
IF l = K THEN m = i - 1 ELSE m = 0
PRINT "m="; m
END
```

Задача 26, стр. 49 (Turbo Pascal)

```
Program Shift_in_array;
Uses CRT;
Const n1=10;
Var
  n,i:integer;
  A:array[1..n1] of real;
  c:real;
Begin
  ClrScr;
  Write('Введите n ');
  Readln(n);
  For i:=1 to n do
    begin
      Write('Введите A[',i,'] ');
      Readln(A[i])
    end;
  c:=A[n];
  For i:=n downto 2 do
    A[i]:=A[i-1];
  A[1]:=c;
  For i:=1 to n do
    Writeln('A[',i,']=',A[i])
End.
```

```
  c:=a[n]
  нц для i от n до 2 шаг -1 |  команда
  |  a[i]:=a[i-1]           |повторения "для"
  кц
  |после сдвига всех элементов массива
  |вправо на одну позицию, a[1]
  a[1]:=c   |получает значение c или a[n]
  |вывод элементов нового массива a
  |организует система Ершол
кон
```

Задача 26, стр. 49 (QBasic)

```
REM Сдвиг в массиве
DIM n AS INTEGER, i AS INTEGER
DIM c AS SINGLE
CLS
INPUT "Введите n "; n
DIM A(1 TO n) AS SINGLE
FOR i = 1 TO n
  PRINT "Введите A("; i; ") ";
  INPUT A(i)
NEXT i
c = A(n)
FOR i = n TO 2 STEP -1
  A(i) = A(i - 1)
NEXT i
A(1) = c
FOR i = 1 TO n
  PRINT "A("; i; ")="; A(i)
NEXT i
END
```

Задача 27, стр. 50 (Ершол, способ 1)

```
алг максимальные в массиве (арг цел n,
                            вещ таб a[1:n])
 нач вещ b, цел i, цел K, цел таб M[1:n]
  |предположим, что больший
  b:=a[1]                 |элемент a[1]
  |будем сравнивать больший элемент с
  |элементами массива, начиная со
  i:=2                    |второго (i=2)
  нц       |команда цикла с постусловием
   |команда ветвления в сокращенной
   если b<=a[i]           |форме для
    то      |выбора большего из b и a[i]
     b:=a[i]
```

Задача 27, стр. 50 (QBasic, способ 1)

```
REM Максимальные элементы
массива1
DIM n AS INTEGER, i AS INTEGER
DIM K AS INTEGER
DIM b AS SINGLE
CLS
INPUT "Введите n "; n
DIM a(1 TO n) AS SINGLE
DIM M(1 TO n) AS INTEGER
FOR i = 1 TO n
  PRINT "Введите A("; i; ") ";
  INPUT a(i)
NEXT i
b = a(1)
i = 2
DO
  IF b <= a(i) THEN b = a(i)
  i = i + 1
LOOP UNTIL i > n
K = 0 : i = 1
DO
  IF b = a(i) THEN
          K = K + 1 : M(K) = i
  END IF
  i = i + 1
LOOP UNTIL i > n
FOR i = 1 TO K
  PRINT "M("; i; ")="; M(i)
NEXT i
PRINT "K="; K
END
```

Задача 27, стр. 50 (Ершол, способ 2)

```
алг максимальные в массиве (арг цел n,
                            арг вещ таб a[1:n])
 нач вещ b, цел i, цел K, цел таб M[1:n]
  |предположим, что больший
   b:=a[1]                |элемент a[1]
  |начальное значение индекса
  |просматриваемого элемента массива
   i:=2                              |равно 2
```

```
    все
   |увеличение индекса просматривае-
    i:=i+1    |мого элемента массива на 1
  кц при  i>n
 |задание начального значения
 |K(количество
  K:=0 |максимальных элементов)
  i:=1 |задание начального значения i
  нц |команда повторения
       |с постусловием
     |команда ветвления в
    если b=a[i]     |сокращенной форме
      то       |подсчет элементов массива,
        K:=K+1 |равных b, создание
       |массива индексов этих элементов
        M[K]:=i
    все
   |получение индекса
    i:=i+1   |просматриваемого элемента
 |проверка условия окончания цикла
  кц при  i>n
  |блок вывода значения K и значений
  |элементов массива M[1:K]
  вывод нс, K
  вывод нс
  нц для i от 1 до K
    вывод M[i]
  кц
кон
```

Задача 27, стр. 50 (Turbo Pascal, способ 1)

```
Program Maxi-
mum_elements_of_the_array;
Uses CRT;
Const n1=10;
Var
  n,i,K:integer;
  a:array[1..n1] of real;
  M:array[1..n1] of byte;
  b:real;
Begin
  ClrScr;
  Write('Введите n '); Readln(n);
  For i:=1 to n do
```

```
 |предположив, что больший элемент
  K:=1       |уже есть (a[1]), имеем K=1
 |предположив, что больший элемент
  M[K]:=1          |a[1], имеем M[1]=1
  нц |команда цикла с постусловием
     |команда ветвления в
      если b<=a[i]   |сокращенной форме
        то |если b>a[i], то больший эле-
         |мент b, если b<=a[i], то рассмот-
         |рим два случая b=a[i] и b<a[i]:
          если b=a[i] |если b=a[i], то a[i]
         |надо сосчитать как очередной
           то            |больший элемент,
            K:=K+1  |увеличим K на 1,
           |элементу массива M[K]
            M[K]:=i |присвоим значение i;
           иначе   |если b<a[i], то большее
           |число a[i],
           |b присвоим значение a[i],
            b:=a[i]
           |количество наибольших из
           |просмотренных элементов
            K:=1                |равно 1,
           |элементу массива M[1]
            M[K]:=i |присвоим значение i
        все
      все
     |индекс очередного
     |просматриваемого элемента
      i:=i+1  |увеличим на 1
  |проверим условие окончания цикла
   кц при  i>n
   |блок вывода значения K и элементов
   |массива M[1:K]
   вывод нс, K
   вывод нс
   нц для i от 1 до K
     вывод M[i]
   кц
кон
```

Задача 27, стр. 50 (Turbo Pascal, способ 2)

```
Program Maxi-
mum_elements_of_the_array;
```

```
  begin
    Write('Введите A[',i,'] '); Readln(a[i])
  end;
 b:=a[1]; i:=2;
 Repeat
  If b<=a[i] then b:=a[i];
  i:=i+1
 Until i>n;
 K:=0;
 i:=1;
 Repeat
  If b=a[i] then
            begin
              K:=K+1; M[K]:=i
            end;
  i:=i+1
 Until i>n;
 For i:=1 to K do
  Writeln('M[',i,']=',M[i]);
 Writeln('K=',K)
End.
```

Задача 27, стр. 50 (QBasic, способ 2)

```
REM Максимальные элементы в масси-
ве
DIM n AS INTEGER, i AS INTEGER
DIM K AS INTEGER, b AS SINGLE
CLS
INPUT "Введите n "; n
DIM a(1 TO n) AS SINGLE
DIM M(1 TO n) AS INTEGER
FOR i = 1 TO n
 PRINT "Введите A("; i; ") ";
 INPUT a(i)
NEXT i
b = a(1) : i = 2
K = 1 : M(K) = 1
DO
 IF b <= a(i) THEN
            IF b = a(i) THEN
                    K = K + 1: M(K) = i
                  ELSE
                    b = a(i)
                    K = 1 : M(K) = i
            END IF
```

```
Uses CRT;
Const n1=10;
Var
 n,i,K:integer;
 a:array[1..n1] of real;
 M:array[1..n1] of byte;
 b:real;
Begin
 ClrScr;
 Write('Введите n ');
 Readln(n);
 For i:=1 to n do
  begin
    Write('Введите A[',i,'] ');
    Readln(a[i])
  end;
 b:=a[1];
 i:=2;
 K:=1;
 M[K]:=1;
 Repeat
  If b<=a[i] then
                  if b=a[i] then
                       begin
                         K:=K+1;
                         M[K]:=i
                       end
                     else
                       begin
                         b:=a[i];
                         K:=1;
                         M[K]:=i
                       end;
  i:=i+1
 Until i>n;
 For i:=1 to K do
  Writeln('M[',i,']=',M[i]);
 Writeln('K=',K)
End.
```

Задача 28, стр. 52 (QBasic)

```
REM Дружественные числа
DIM k AS INTEGER, n AS INTEGER
DIM p AS INTEGER, s AS INTEGER
CLS
```

```
  END IF
  i = i + 1
LOOP UNTIL i > n
FOR i = 1 TO K
  PRINT "M("; i; ")="; M(i)
NEXT i
PRINT "K="; K
END
```

Задача 28, стр. 52 (Ершол)

```
алг дружественные числа (арг цел n)
 дано
 надо
нач цел k,p,s, цел таб Делитель[1:n]
  |первое просматриваемое число
  |равно 2
   k:=2
  |цикл "пока", проверка условия
   нц пока k<=n            |окончания цикла
     |единица - делитель всех просматри-
      Делитель[k]:=1 |ваемых чисел
     |очередное просматриваемое число
      k:=k+1
   кц
  |первое просматриваемое число
  |равно 2
   k:=2
  |цикл "пока", проверка условия
   нц пока k<=div(n,2) |окончания цикла
     |первое число имеющее делителем
      p:=k+k |число k (p<>k)
     |цикл "пока", проверка условия
      нц пока p<=n |окончания цикла
        |увеличение суммы делителей
        |y чисел кратных k (p<>k)
         Делитель[p]:=Делитель[p]+k
        |следующее число имеющее
         p:=p+k |делителем число k
      кц
     |очередное просматриваемое число
      k:=k+1
   кц
  |организация поиска дружественных
  |чисел
   нц для k от 2 до n-1
```

```
INPUT "Введите n "; n
DIM Divisor(1 TO n)  AS INTEGER
k = 2
WHILE k <= n
  Divisor(k) = 1
  k = k + 1
WEND
k = 2
WHILE k <= n \ 2
  p = k + k
  WHILE p <= n
    Divisor(p) = Divisor(p) + k
    p = p + k
  WEND
  k = k + 1
WEND
FOR k = 2 TO n - 1
  FOR s = k + 1 TO n
IF (Divisor(k)=s)AND(Divisor(s)=k)
THEN
     PRINT k, s
END IF
  NEXT s
NEXT k
END
```

Задача 29, стр. 53 (Turbo Pascal)

```
Program Grouping;
Uses CRT;
Var
  n,i,k,j:integer;
  a:array [1..20] of real;
  c:real;
Begin
  ClrScr;
  Write('Введите n ');
  Readln(n);
  For i:=1 to n do
    begin
      Write('Введите a[',i,'] ');
      Readln(a[i])
    end;
  i:=1;
  k:=0;
  While k<n do
```

```
    нц для s от k+1 до n
      если s=Делитель[k] и
                          k=Делитель[s]
        то
          вывод нс, k, s
      все
    кц
  кц
кон
```

Задача 28, стр. 52 (Turbo Pascal)

```
Program Amicable_numbers;
Uses CRT;
Const n1=20000;
Var
  k,n,p,s:integer;
  Divisor:array [1..n1] of integer;
Begin
  Write('Введите n '); Readln(n);
  k:=2;
  While k<=n do
    begin
      Divisor[k]:=1;
      k:=k+1
    end;
  k:=2;
  While k<=(n div 2) do
    begin
      p:=k+k;
      While p<=n do
        begin
          Divisor[p]:=Divisor[p]+k;
          p:=p+k
        end;
      k:=k+1;
    end;
  For k:=2 to n-1 do
    For s:=k+1 to n do
      If (Divisor[k]=s) and (Divisor[s]=k)
then
      Writeln('Дружественные числа ',k,' и
',s)
End.
```

```
    begin
      If A[i]>0 then
              i:=i+1
            else
              begin
                c:=A[i];
                j:=i+1;
                While j<=n do
                  begin
                    A[j-1]:=A[j];
                    j:=j+1
                  end;
                A[n]:=C
              end;
      k:=k+1
    end;
  For i:=1 to n do
    Writeln('A[',i,']=',A[i])
End.
```

Задача 30, стр. 57 (QBasic)

```
REM Наибольший общий делитель
DECLARE SUB NOD (m AS INTEGER,
          n AS INTEGER, t AS INTEGER)
DIM a AS INTEGER, b AS INTEGER
DIM c AS INTEGER, d AS INTEGER
DIM z AS INTEGER
INPUT "Введите a,b,c,d "; a, b, c, d
NOD (a), (b), z
NOD (z), (c), z
NOD (z), (d), z
PRINT "НОД(";a;",";b;",";c;",";d;")="; z
END

SUB NOD (m AS INTEGER,
         n AS INTEGER, t AS INTEGER)
WHILE m <> n
IF m>n THEN m = m - n ELSE n = n - m
WEND
t = m
PRINT t
END SUB
```

Задача 29, стр. 53 (Ершол)

```
алг группировка (арг цел n,
                 арг рез вещ таб A[1:n])
нач цел i,k,j, вещ C
 |задание начального значения индекса
  i:=1 |просматриваемого элемента
  k:=0|количество просмотренных
       |элементов
 |команда цикла с предусловием,
 |проверка условия окончания цикла
  нц пока k<n
   | если A[i]>0 |если A[i]>0, то этот
   |  то |элемент оставляем на месте,
   |     |получаем индекс следующего
   |   i:=i+1|просматриваемого элемента
   |  иначе  |если A[i]<0, то организуем
   |   C:=A[i] |сдвиг на 1 элемент влево,
   |   j:=i+1    |начиная с A[i+1] до A[n],
   |  |A[i] ставим на место A[n]
   |   нц пока j<=n
   |   | A[j-1]:=A[j] |получим
   |   | j:=j+1      |последовательность
   |   кц            |A[1], A[2], ... , A[i-1],
   |   A[n]:=C     |A[i+1], ... , A[n], A[i]
   |  все
   |  k:=k+1
  кц
кон
```

Задача 29, стр. 53 (QBasic)

```
REM Группировка
DIM n AS INTEGER, i AS INTEGER
DIM k AS INTEGER, j AS INTEGER
DIM C AS SINGLE
CLS
INPUT "Введите n "; n
DIM A(1 TO n) AS SINGLE
FOR i = 1 TO n
  PRINT "Введите a("; i; ") ";
  INPUT A(i)
NEXT i
i = 1
k = 0
WHILE k < n
  IF A(i) > 0 THEN
```

Задача 31, стр. 58 (Ершол)

```
алг Мерсенн (арг цел n)
 нач цел n1,p,Кандидат, лог fl,fl1
  |n1 - целая часть от деления n на 2
  n1:=div(n,2)
  |просмотрим все значения p от 2 до
  |k, чтобы исследовать все числа вида
  |2^p-1<=n на принадлежность числам
  |Мерсенна
  p:=2
  |первый кандидат в числа Мерсенна -
  Кандидат:=3          |3=2^2-1
 |fl=нет - очередное исследуемое
  fl:=нет   |число 2^p-1<=n, fl=да -
  |очередное исследуемое
  |число 2^p-1>n
  |команда повторения с постусловием
  нц
   |команда обращения к
   | вспомогательному алгоритму для
   | проверки, является ли число p
   простое(p,fl1)       |простым,
   если fl1 |если число p - простое
   |  то       |(fl1=да), то
   |   |2^p-1 - число Мерсенна
   |   |вывод очередного числа
   |   вывод нс, Кандидат     |Мерсенна
   все
   |получение очередного
   p:=p+1       |исследуемого числа p
   |получение очередного
   если Кандидат<=n1 |исследуемого
   |  то    |кандидата в числа Мерсенна
   |   Кандидат:=Кандидат*2
   |   если Кандидат<n
   |   | то Кандидат:=Кандидат+1
   |   | иначе fl:=да
   |   все
   |  иначе
   |   fl:=да
   все
  кц при fl
кон
|вспомогательный алгоритм
```

```
          i = i + 1
        ELSE
          C = A(i)
          j = i + 1
          WHILE j <= n
            A(j - 1) = A(j)
            j = j + 1
          WEND
          A(n) = C
  END IF
  k = k + 1
WEND
FOR i = 1 TO n
  PRINT "A("; i; ")="; A(i)
NEXT i
END
```

Задача 30, стр. 57 (Ершол)

```
алг НОД4(арг цел a,b,c,d, рез цел z)
нач
| НОД(a,b,z) |команды обращения к
| НОД(z,c,z) |вспомогательному
| НОД(z,d,z) |алгоритму
кон
|вспомогательный алгоритм вычисления
|НОД двух чисел
алг НОД(арг цел m,n, рез цел t)
нач цел m1,n1 |при хорошем стиле
|программирования значения аргументов
|в программе менять нельзя
| m1:=m; n1:=n
| |команда цикла с предусловием
| нц пока m1<>n1
| | |команда ветвления в полной форме
| | если m1>n1
| | | то  m1:=m1-n1     |большее из двух
| | |                   |чисел заменяем
| | | иначе n1:=n1-m1 |разностью
| | все                 |этих чисел
| кц
| t:=m1
кон
```

Задача 30, стр. 57 (Turbo Pascal)

```
Program Greatest_common_divisor;
Uses CRT;
```

```
алг простое(арг цел a, рез лог fl1)
 нач цел k
 | |очередной возможный делитель
 | k:=2                        |числа a
 | |предположим, что число
 | fl1:=да                     |простое fl1=да
 | |цикл "пока", проверка условия
 | нц пока k<=div(a,2) и fl1 |окончания
 |                             |цикла
 | | если mod(a,k)=0         |если число a
 | | | то fl1:=нет           |делится на k без
 | | | |остатка, то число a не является
 | | | |простым (fl1=нет)
 | | все
 | |получение очередного возможного
 | | k:=k+1                  |делителя числа a
 | кц
кон
```

Задача 31, стр. 58 (Turbo Pascal)

```
Program Mersenn_numbers;
Uses CRT;
Var
  n,n1,p,Candidate:integer;
  fl,fl1:boolean;
Procedure Prime(a:integer;Var
fl1:boolean);
Var
  k:integer;
Begin
  k:=2;
  fl1:=true;
  While (k<=a div 2) and (fl1) do
    begin
      If a mod k=0 then
                        fl1:=false;
      k:=k+1
    end;
  Writeln(fl1)
End;
Begin
  ClrScr;
  Write('Введите n ');
  Readln(n);
  n1:=n div 2;
```

```
Var
  a,b,c,d,z:integer;
Procedure NOD(m,n:integer; Var
t:integer);
Begin
  While m<>n do
    If m>n then m:=m-n else n:=n-m;
  t:=m;
  Writeln(t)
End;
Begin
  ClrScr;
  Write('Введите a,b,c,d '); Readln(a,b,c,d);
  NOD(a,b,z);
  NOD(z,c,z);
  NOD(z,d,z);
  Writeln('НОД(',a,',',b,',',c,',',d,')=',z)
End.
```

Задача 31, стр. 58 (QBasic)

```
REM Числа Мерсенна
DECLARE SUB Prime (a AS INTEGER,
fl1 AS INTEGER)
DIM n AS INTEGER, n1 AS INTEGER
DIM p AS INTEGER
DIM Candidate AS INTEGER
DIM fl  AS INTEGER, fl1 AS INTEGER
INPUT "Введите n "; n
n1 = n \ 2
p = 2
Candidate = 3
fl = 0
DO
  Prime (p), fl1
  IF fl1 = 1 THEN
    PRINT "Число Мерсенна - "; Candidate
  END IF
  p = p + 1
  IF Candidate <= n1 THEN
     Candidate = Candidate * 2
     IF Candidate < n THEN
                    Candidate = Candidate + 1
                         ELSE
                      fl = 1
     END IF
  p:=2;
  Candidate:=3;
  fl:=false;
  Repeat
    Prime(p,fl1);
    If fl1 then
  Writeln('Число Мерсенна-', Candidate);
    p:=p+1;
    If Candidate<=n1 then
                  begin
                    Candidate:=Candidate*2;
                    If Candidate<n then
                      Candidate:=Candidate+1
                                  else
                        fl:=true
                  end
                 else
                    fl:=true
  Until fl;
End.
```

Задача 32, стр. 60 (Ершол)

```
алг больший в массиве (арг цел n,
              арг вещ таб a[1:n], рез вещ t)
 нач цел k
 | |индекс очередного
  k:=2      |просматриваемого элемента
 |предположим, что больший
  t:=a[1]                  |элемент a[1]
 |цикл с предусловием, проверка
  нц пока k<=n |условия окончания
                 |цикла
 | | |обращение к вспомогательному
 | | |алгоритму для выбора большего из
 | | |двух: t и очередного просматри-
 | | |ваемого элемента массива
 | | БИД(t,a[k],t)
 | | |индекс очередного просматривае-
 | |  k:=k+1 |мого элемента
 | кц
кон
|вспомогательный алгоритм
|нахождения большего из m и
|l, и присвоение большего
|значения переменной z
```

```
                ELSE
    fl = 1
  END IF
LOOP UNTIL fl = 1
END

SUB Prime (a AS INTEGER,
           fl1 AS INTEGER)
DIM k AS INTEGER
k = 2
fl1 = 1
WHILE (k <= a \ 2) AND (fl1 = 1)
  IF a MOD k = 0 THEN fl1 = 0
  k = k + 1
WEND
PRINT fl1
END SUB
```

Задача 32, стр. 60 (Turbo Pascal)

```
Program Most_element_in_array;
Uses CRT;
Var
  n,k:integer;
  a:array [1..10] of real;
  t:real;
Procedure BID(m,l:real;Var z:real);
Begin
  If m>l then z:=m else z:=l;
  Writeln(z)
End;
Begin
  ClrScr;
  Write('Введите n '); Readln(n);
  For k:=1 to n do
    begin
      Write('Введите a[',k,'] '); Readln(a[k])
    end;
  k:=2; t:=a[1];
  While k<=n do
    begin
      BID(t,a[k],t);
      k:=k+1
    end;
  Writeln('Максимум ',t)
End.
```

```
алг БИД(арг вещ m,l, рез вещ z)
 нач
  если m>l
    то z:=m
    иначе z:=l
  все
кон
```

Задача 32, стр. 60 (QBasic)

```
REM Наибольший элемент в массиве
DECLARE SUB BID (m AS SINGLE,
           l AS SINGLE, z AS SINGLE)
DIM n AS INTEGER, k AS INTEGER
DIM t AS SINGLE
CLS
INPUT "Введите n "; n
DIM a(1 TO n) AS SINGLE
FOR k = 1 TO n
  PRINT "Введите a("; k; ") ";
  INPUT a(k)
NEXT k
k = 2
t = a(1)
WHILE k <= n
  BID (t), (a(k)), t
  k = k + 1
WEND
PRINT "Максимум "; t
END

SUB BID (m AS SINGLE, l AS SINGLE,
           z AS SINGLE)
IF m > l THEN z = m ELSE z = l
PRINT z
END SUB
```

Задача 33, стр. 61 (Turbo Pascal)

```
Program Twin_primes;
Uses CRT;
Var
  n,p:integer;
  fl:boolean;
Procedure Prime(a:integer;Var
fl1:boolean);
Var
  k:integer;
```

Задача 33, стр. 61 (Ершол)

```
алг близнецы(арг цел n)
 нач цел p, лог fl
  p:=2|очередное просматриваемое число
  |цикл "пока",
   нц пока p<=n-2    |проверка условия
                     |p<=n-2
     |обращение к вспомогательному
     |алгоритму для проверки является ли
      простое(p,fl)       |число p простым,
      если fl |если число p простое (fl=да),
        то  |то проверим является ли число
         простое(p+2,fl)     |p+2 простым,
         если fl |если число p+2 простое
                 |(fl=да),
           то   |то p и p+2 - близнецы,
            |выводим их на экран
            вывод нс,p,p+2
           все
        все
      p:=p+1
  кц
кон
|вспомогательный алгоритм определения
|является ли число p простым (fl1=да)
алг простое(арг цел a, рез лог fl1)
 нач цел k   |очередной возможный
             |делитель числа a
  если a<>2        |если a<>2,
    то             |исследуем это число,
     если mod(a,2)<>0 |если число
       то   |нечетное, mod(a,2)<>0, то
            |исследуем это число
        |делителями нечетного числа
        |могут быть только нечетные
        |числа k<=div(a,2)+1
        |k - целая часть от деления a на 2
        k:=div(a,2)
        если mod(k,2)=0   |если k -
                          |четное число,
          то      |то получим следующее за
           k:=k+1     |ним нечетное число
        все
        нц пока mod(a,k)<>0   |в цикле
```

```
Begin
 If a<>2 then
       If a mod 2 <>0 then
                  begin
                   k:=a div 2;
                   If k mod 2=0 then
                                 k:=k+1;
                   While a mod k<> 0 do
                    k:=k-2;
                   If k=1 then fl1:=true
                          else fl1:=false
                  end
                 else fl1:=false
      else
       fl1:=true;
 Writeln(fl1)
End;
Begin
 ClrScr;
 Write('Введите n '); Readln(n);
 p:=2;
 While p<=n-2 do
  begin
   Prime(p,fl);
   If fl then
        begin
         Prime(p+2,fl);
         If fl then
           Writeln('Близнецы: ',p,';',p+2)
        end;
   p:=p+1
  end
End.
```

Задача 33, стр. 61 (QBasic)

```
REM Близнецы
DECLARE SUB Prime (a AS INTEGER,
                   fl1 AS INTEGER)
DIM n AS INTEGER, p AS INTEGER
DIM fl  AS INTEGER
CLS
INPUT "Введите n "; n
p = 2
WHILE p <= n-2
 Prime (p), fl
```

```
|  |  | k:=k-2|"пока" ищем из нечетных
|  |  кц|делителей числа a больший,
|  |  если k=1  |если наибольший
|  |  | |делитель числа a равен 1,
|  |  | то fl1:=да   |то число простое
|  |  |              |(fl1=да),
|  |  | иначе fl1:=нет  |иначе число не
|  |  все    |является простым (fl1=нет),
|  | иначе   |если число четное, то оно
|  | fl1:=нет  |не является простым
|  все          |fl1=нет
| иначе       |если a=2,
| fl1:=да     |то число простое – fl1=да
| все
кон
```

Задача 34, стр. 62 (Ершол)

```
алг сумма Фибоначчи(арг цел m)
 дано
 надо
нач цел s,f
| s:=m |текущей сумме присвоим
|      | значение m
| нц пока s>0 |пока s>0 обращаемся к
| |вспомогательному алгоритму для
| |нахождения f - наибольшего числа
| Фиб(s,f)          |Фибоначчи <= s
| s:=s-f |уменьшаем текущую сумму s
|        |на f
| вывод f, " +"
| кц
кон
|вспомогательный алгоритм для
|нахождения наибольшего числа
|Фибоначчи <=m1

алг Фиб(арг цел m1, рез цел c)
|значение найденного числа Фибоначчи
|присвоим c
 нач цел a,b
| a:=1;b:=1;c:=1
| нц пока c<m1
| | c:=a+b
| | a:=b
| | b:=c
```

```
 IF fl = 1 THEN
          Prime (p + 2), fl
          IF fl = 1 THEN
            PRINT "Близнецы: "; p; p + 2
          END IF
 END IF
 p = p + 1
WEND
END

SUB Prime (a AS INTEGER,
           fl1 AS INTEGER)
DIM k AS INTEGER
IF a <> 2 THEN
  IF a MOD 2 <> 0 THEN
     k = a \ 2
     IF k MOD 2 = 0 THEN k = k + 1
     WHILE a MOD k <> 0
        k = k - 2
     WEND
     IF k = 1 THEN fl1 = 1 ELSE fl1 = 0
  ELSE
     fl1 = 0
   END IF
ELSE
   fl1 = 1
END IF
PRINT fl1
END SUB
```

Задача 35, стр. 62 (Ершол)

```
алг  Сдвиг1(арг цел k,n,
            арг рез вещ таб B[1:n])
 нач цел i |i - номер очередного сдвига
           |на 1 элемент
| |организуем k сдвигов элементов мас-
| нц для i от 1 до k |сива на 1 элемент
| | |обращение к вспомогательному
| | |алгоритму для очередного сдвига
| | сдвиг(n,B)          |на 1 элемент
| кц
кон
|вспомогательный алгоритм,
|производящий сдвиг элементов
|массива A на 1 элемент вправо
```

```
  кц
  если c>m1
  | то
  |  c:=a
  все
кон
```

Задача 34, стр. 62 (QBasic)

```
REM Сумма чисел Фибоначчи
DECLARE SUB Fib (m1 AS INTEGER,
                 c AS INTEGER)
DIM m AS INTEGER, s AS INTEGER
DIM f AS INTEGER
CLS
INPUT "Введите m "; m
s = m
WHILE s > 0
  Fib (s), f
  s = s - f
  PRINT f;
WEND
END

SUB Fib (m1 AS INTEGER,
         c AS INTEGER)
DIM a AS INTEGER, b AS INTEGER
a = 1 : b = 1 : c = 1
WHILE c < m1
  c = a + b: a = b: b = c
WEND
IF c > m1 THEN c = a
END SUB
```

Задача 34, стр. 62 (Turbo Pascal)

```
Program Sum_of_Fibonacci_numbers;
Uses CRT;
Var
  m,s,f:integer;
Procedure Fib(m1:integer; Var c:integer);
Var
  a,b:integer;
Begin
  a:=1; b:=1; c:=1;
  While c<m1 do
    begin
```

```
алг сдвиг(арг цел n,
          арг рез вещ таб A[1:n])
 нач вещ c, цел i
  c:=A[n]
  нц для i от n до 2 шаг -1
  |  A[i]:=A[i-1]
  кц
  A[1]:=c
кон
```

Задача 35, стр. 62 (Turbo Pascal)

```
Program Shift1_element_of_array;
Uses CRT;
Const n1=10;
Type
  mass=array[1..n1] of real;
Var
  i,k,n:integer;
  B:mass;
Procedure Shift(n:integer; Var A:mass);
Var
  c:real;
  i:integer;
Begin
  c:=A[n];
  For i:=n downto 2 do
    A[i]:=A[i-1];
  A[1]:=c
End;
Begin
  ClrScr;
  Write('Введите k – величину сдвига');
  Readln(k);
  Write('Введите n ');
  Readln(n);
  For i:=1 to n do
    begin
      Write('Введите B[',i,'] ');
      Readln(B[i])
    end;
  For i:=1 to k do
    Shift(n,B);
  For i:=1 to n do
    Writeln('B[',i,']=',B[i])
End.
```

```
    c:=a+b; a:=b; b:=c
   end;
  If c>m1 then c:=a
End;
Begin
  ClrScr;
  Write('Введите m '); Readln(m);
  s:=m;
  While s>0 do
   begin
    Fib(s,f);
    s:=s-f;
    Write(f,'+')
   end;
End.
```

Задача 35, стр. 62 (QBasic)

```
REM Сдвиг1 элементов в массиве
DECLARE SUB Shift (n AS INTEGER,
                   A() AS SINGLE)
DIM i AS INTEGER, k AS INTEGER
DIM n AS INTEGER
CLS
INPUT "Введите k – величину сдвига"; k
INPUT "Введите n "; n
DIM B(1 TO n) AS SINGLE
FOR i = 1 TO n
  PRINT "Введите B("; i; ")";
  INPUT B(i)
NEXT i
FOR i = 1 TO k
  Shift (n), B()
NEXT i
FOR i = 1 TO n
  PRINT "B("; i; ")="; B(i)
NEXT i
END
SUB Shift (n AS INTEGER,
           A() AS SINGLE)
DIM c AS SINGLE, i AS INTEGER
c = A(n)
FOR i = n TO 2 STEP -1
  A(i) = A(i - 1)
NEXT i
A(1) = c
```

Задача 36, стр. 63 (Ершол)

```
алг Кратные пары(арг цел n,
  арг цел таб A[1:n],B[1:n], рез цел k,s)
 нач цел i,r,q
 | k:=0|текущее количество кратных пар
 | |текущая сумма частных от деления
 |  s:=0       |большего числа на меньшее
 | |индекс очередной рассматриваемой
 | |пары элементов из массивов A и B
 |  i:=1
 | |цикл "пока",
 |  нц пока i<=n          |проверка условия
 |  | |обращение к вспомогательному
 |  | |алгоритму для нахождения
 |  | |частного и остатка от деления
 |  | |рассматриваемых элементов пары
 |  |  Делимость(A[i],B[i],q,r)
 |  |  если r=0 |если текущая пара - пара
 |  |  | то       |кратных чисел, то
 |  |  |  k:=k+1 |увеличиваем число
 |  |  |  |кратных пар на 1,
 |  |  |  |текущую сумму увеличиваем
 |  |  |  s:=s+q  |на полученное частное q
 |  |  все
 |  | |индекс следующей рассматри-
 |  |  i:=i+1 |ваемой пары элементов
 | кц
кон
|вспомогательный алгоритм
|определения остатка и частного от
|деления большего числа из чисел
|x и y на меньшее из них
алг Делимость(арг цел x,y, рез цел q,r)
 нач
 | если x>y
 |  то
 |  q:=div(x,y)
 |  r:=mod(x,y)
 |  иначе
 |  q:=div(y,x)
 |  r:=mod(y,x)
 | все
кон
```

Задача 36, стр. 63 (QBasic)

```
END SUB
```

Задача 36, стр. 63 (Turbo Pascal)

```
Program Multiple_pairs;
Uses CRT;
Const n1=10;
Var
  i,k,n,s,q,r:byte;
  A,B:array[1..n1] of byte;
Procedure Divisibility(x,y:byte;
                       Var q,r:byte);
Begin
  If x>y then
          begin
            q:=x div y; r:=x mod y
          end
        else
          begin
            q:=y div x; r:=y mod x
          end
End;
Begin
  ClrScr;
  Write('Введите n '); Readln(n);
  For i:=1 to n do
    begin
      Write('Введите A[',i,'] '); Readln(A[i])
    end;
  For i:=1 to n do
    begin
      Write('Введите B[',i,'] '); Readln(B[i])
    end;
  k:=0; s:=0; i:=1;
  While i<=n do
    begin
      Divisibility(A[i],B[i],q,r);
      If r=0 then
              begin
                k:=k+1; s:=s+q
              end;
      i:=i+1
    end;
  Writeln('k=',k,' s=',s)
End.
```

```
REM Кратные пары
DECLARE SUB Divisibility (x AS
INTEGER, y AS INTEGER, q AS
INTEGER, r AS INTEGER)
DIM i AS INTEGER, k AS INTEGER
DIM n AS INTEGER, s AS INTEGER
DIM q AS INTEGER, r AS INTEGER
CLS
INPUT "Введите n "; n
DIM A(1 TO n) AS INTEGER
DIM B(1 TO n) AS INTEGER
FOR i = 1 TO n
  PRINT "Введите A("; i; ")";
  INPUT A(i)
NEXT i
FOR i = 1 TO n
  PRINT "Введите B("; i; ")";
  INPUT B(i)
NEXT i
k = 0
s = 0
i = 1
WHILE i <= n
  Divisibility (A(i)), (B(i)), q, r
  IF r = 0 THEN
           k = k + 1
           s = s + q
  END IF
  i = i + 1
WEND
PRINT "k="; k; "s="; s
END

SUB Divisibility (x AS INTEGER,
y AS INTEGER, q AS INTEGER,
r AS INTEGER)
IF x > y THEN
          q = x \ y
           r = x MOD y
        ELSE
          q = y \ x
          r = y MOD x
END IF
END SUB
```

3. Занимательные игры-алгоритмы

Игры и головоломки – восхитительный мир для развития алгоритмического мышления, составления алгоритмов и компьютерных моделей.

Уточнение понятия «алгоритм» полезно организовать в игровых микромирах. Мы рекомендуем для занятий по теме «Алгоритм. Свойства алгоритма. Способы записи алгоритма» использовать игры: «Жизнь», «Баше», «Ханойская башня» и пр. Они интересны, доступны для понимания; при составлении игровых алгоритмов, обучаемые уточняют понятия «алгоритм», «исполнитель». Игровые алгоритмы вводят их в класс базовых управляющих команд (цепочка, ветвление, повторение), которые они будут применять в дальнейшем для записи алгоритмов решения различных задач, они познакомятся с понятием «вспомогательный алгоритм». В процессе обсуждения алгоритма игры учащиеся встречаются с различными типами данных (логическими, символьными, литерными, числовыми), рассматривают методы проектирования алгоритмов «сверху-вниз» и «снизу-вверх».

Огромным познавательным потенциалом обладают логические игры. Увлекательный мир логических игр способствует развитию познавательных способностей обучаемых, повышает эффективность изучения соответствующей темы. Программирование игр позволяет реализовать имеющиеся знания в реальный продукт. Игровой элемент создает неформальную обстановку, заинтересованность и тем самым стимулирует преодоление трудностей в написании своей игры-алгоритма или в анализе представленного алгоритма.

В нашей работе представлены примеры записи игровых алгоритмов: блок-схемы, коды алгоритмов на языках QBasic и Turbo Pascal. Они реализуют ядро в программировании каждой игры. Желательно, чтобы читатель видоизменил программу, в соответствии с

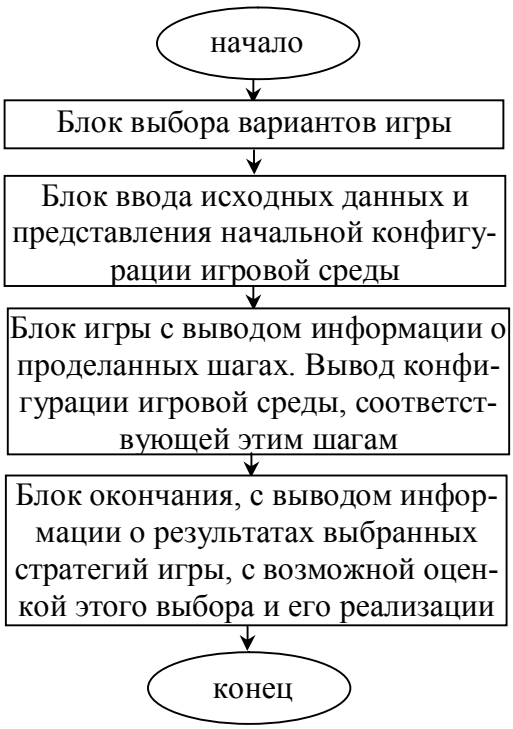
особенностями используемой для программирования системы, личными знаниями и вкусами.

Структурная схема возможных блоков построения игровых алгоритмов

Можно выделить следующие компоненты игровой программы: блок описания игры и представления начальной конфигурации игровой среды; основной блок; блок окончания игры. Структуру данных определяют на начальной стадии разработки игровой программы.

Начальный блок. В начальном блоке игровой программы реализуется вывод на экран названия игры, ее краткой истории, имени автора созданной игровой программы, даты создания программы, вопрос о том, нужно ли вывести на экран правила игры (при утвердительном ответе программа должна выдать объяснительный текст, в противном случае, игра переходит на следующую стадию). В начальном блоке программируется выбор вариантов игры, если их несколько.

Схема возможных блоков построения игровой программы



Основной блок. Это та часть программы, где, собственно, и происходит игра. Здесь реализуются изменения внутренних структур данных, вывод на экран конфигурации игровой среды, соответствующей сделанным в игре шагам. В этой части программы необходимо организовать диалог с игроками для реализации ходов в игре, т.е. восприятие управляющих воздействий, их анализ, изменение внутренних структур данных (по правилам игры), отображение этих изменений на экране дисплея, чтобы игрок мог увидеть результат своих действий в виде новой ситуации. Действия, запрограммированные в основном блоке, повторяются многократно до тех пор, пока не будет достигнуто усло-

вие окончания игры, проверяемое внутри этого блока. Таким условием может быть некоторая конечная конфигурация игровой среды, нажатие игроком определенной клавиши, например, «конец игры», ограничение на число повторений, шагов или ходов игры и др.

Блок окончания. В блоке окончания выводится информация о результатах выбранных стратегий игры с возможной оценкой этого выбора и его реализации.

Рассмотренная структурная схема, конечно, является приблизительной и достаточно сильно варьируется в деталях для конкретных программ.

3.1. Задача Баше

Приведем пример алгоритма принятия решения (игровой стратегии) в игре «Залача Баше».

Задача 1. Игра Баше для 11 предметов. На столе 11 предметов, например, карандашей. Первый играющий может взять 1, 2 или 3 карандаша. Затем второй играющий может взять 1, 2 или 3 карандаша из оставшихся, затем берет первый и т.д. Итак, поочередно, оба играющих берут не более, чем по 3 карандаша. Проигрывает тот, которому приходится взять последний карандаш. Напишите алгоритм выигрышной стратегии начинающего в этой игре игрока.

Решение. Пусть первый игрок А, второй – В. Здесь мы опишем выигрышную стратегию игрока А. Проведем анализ игры. Чтобы выиграл игрок А, он должен оставить игроку В в последнем ходе 1 предмет. Очевидно, что в предыдущем ходе А должен оставить на столе 5 предметов, иначе он проиграет. В третьем ходе от конца А должен оставить 9 предметов, а всего 11 предметов, следовательно, в третьем ходе от конца или в первом ходе А должен взять 2 предмета.

Алгоритм достижения игроком А победы формулируется так:

1. Первый ход. А берет 2 предмета.

2. Очередной ход. Если В в своем ходе берет K карандашей ($1 \le K \le 3$), причем не все карандаши взяты, то А берет $4-K$ карандаша.

Задача 2. Игра Баше для N предметов. Сформулируем задачу в общем виде: имеется n предметов, двое играющих берут по очереди 1, 2 или 3 предмета, проигрывает тот, кто забирает последний предмет,

написать алгоритм выигрышной стратегии в этой игре начинающего игрока.

Решение. Разобьем все возможные первоначальные количества предметов n на 4 класса в зависимости от остатка, полученного от деления на 4. К первому классу отнесем количество предметов n, которые можно представить как $4m$, ко второму классу отнесем количество предметов n, которые можно представить как $4m+1$, к третьему классу отнесем количество предметов n, которые можно представить как $4m+2$, к четвертому классу отнесем количество предметов n, которые можно представить как $4m+3$. Если первоначальное количество предметов представлено как $4m$, то, взяв один предмет, начинающий игрок оставит количество предметов, принадлежащих классу $4m+3$, взяв 2 предмета – $4m+2$, взяв 3 предмета – $4m+1$. Составим соответствующую таблицу перехода из одного класса в другой, в зависимости от взятых в очередной ход предметов.

Количество взятых предметов	Классы предметов, в зависимости от остатка при делении n на 4			
	$4m$	$4m+1$	$4m+2$	$4m+3$
1	$4m+3$	$4m$	$4m+1$	$4m+2$
2	$4m+2$	$4m+3$	$4m$	$4m+1$
3	$4m+1$	$4m+2$	$4m+3$	$4m$

Если первоначальное количество предметов $4m+1$, то это проигрышное количество предметов для начинающего игрока, потому что сколько бы начинающий не взял предметов, противник вернет его в этот же класс. Единица принадлежит этому классу, тогда возникает необходимость в последней раз начинающему взять один последний предмет. Если первоначальное количество предметов $4m$, $4m+2$, $4m+3$, то выигрышная стратегия начинающего игрока, взять в очередной ход столько предметов, чтобы оставить партнеру количество предметов, принадлежащих классу $4m+1$.

Обозначим данные, используемые для составления алгоритмов и программ и выразим зависимость между данными в виде формул. Число предметов – N. Число предметов, взятых первый раз первым игроком (человеком или компьютером) – P, где $P = (N\text{-}1) \bmod 4$ (остаток от деления N-1 на 4). Число предметов, взятых вторым игроком (человеком) – Y. Число предметов, взятых первым игроком (человеком или компьютером) – C. Между значениями переменных имеется зависимость $Y+C = 4$. Число оставшихся предметов после очередных ходов $N\text{-}C\text{-}Y$. Первый и второй игрок берут в сумме 4 предмета, пока остается больше одного предмета.

Таким образом, исходным данным является первоначальное количество предметов N. Для создания более интересной игровой ситуации целесообразно

задавать число *N* случайным образом в некотором разумном, наперед заданном интервале, например, от 10 до 30.

Составим алгоритм и программу выигрышной стратегии для начинающего игрока, если начальное количество предметов *N*. Рассмотрим один из подходов к составлению алгоритма решения поставленной задачи: начинает игру *первый* игрок (человек или компьютер), игра прерывается, если при заданном количестве предметов *N*, первый игрок при правильной игре выиграть не может.

Словесное описание алгоритма игры Баше, если первый ход Ваш и Вы знаете выигрышную стратегию в этой игре:

1 шаг. Попросите Вашего партнера назвать начальное количество предметов *N* (если партнер назвал неверное количество предметов, попросите выбрать количество предметов еще раз) и идите на шаг 2.

2 шаг. Найдите остаток от деления *N*-1 на 4 и присвойте значение остатка переменной *P* (P:=(N-1) mod 4). Идите на шаг 3.

3 шаг. Сравните *P* с 0. Если *P*=0, то сообщите партнеру, что при правильной игре Вы выиграть не можете и идите на шаг 12. Если *P*<>0, то идите на шаг 4.

4 шаг. Сообщите «Я делаю первый ход» и идите на шаг 5.

5 шаг. Переменной *C* (количество предметов, взятых Вами за один ход) присвойте значение *P* (C:=P). Сообщите «Я беру C предметов» и идите на шаг 6.

6 шаг. Переменной *N* присвойте значение *N*-*C* (N:=N-C). Сообщите «Осталось *N* предметов» и идите на шаг 7.

7 шаг. Сравните *N* с 1. Если *N* = 1, то идите на шаг 11. Если *N*>1, то сообщите «Ваш ход» и идите на шаг 8.

8 шаг. Спросите партнера «Сколько предметов Вы берете?». Переменной *Y* присвойте значение количества предметов, взятых партнером (*Y* может равняться 1 или 2, или 3, если партнер взял неверное количество предметов, попросите повторить ход) и идите на шаг 9.

9 шаг. Вычислите 4-*Y*. Сообщите «Я беру 4-*Y* предметов» и идите на шаг 10.

10 шаг. Переменной *N* присвойте значение *N*-4 (N:=N-4). Сообщите «Осталось *N* предметов» и идите на шаг 7.

11 шаг. Сообщите «Ваш ход. Вы проиграли» и идите на шаг 12.

12 шаг. Спросите партнера «Хотите сыграть еще?». Если партнер ответил «Да», то идите на шаг 1, иначе – на шаг 13.

13 шаг. Закончите игру.

Программа на языке QBasic игры Баше

```
REM Игра Баше
DIM n AS INTEGER, p AS INTEGER, c AS INTEGER, y AS INTEGER, w AS
STRING
DO
 CLS
 LOCATE 7, 35: PRINT "Игра Баше"
 LOCATE 10, 20: PRINT "Играйте правильно. Я делаю первый ход"
 DO
  LOCATE 12, 2: INPUT "Количество предметов для игры: "; n
  p = (n - 1) MOD 4
  IF p < 0 THEN
          LOCATE 13, 2: PRINT "Количество предметов - положительно!"
          SLEEP (2)
          LOCATE 12, 2: PRINT SPC(40);
          LOCATE 13, 2: PRINT SPC(40);
  END IF
 LOOP UNTIL p >= 0
 IF p = 0 THEN
         PRINT "При правильной игре, ";
         PRINT "я у Вас выиграть не могу"
        ELSE
         PRINT "Я делаю первый ход"
         c = p: PRINT "Я беру "; c; " предм., ";
         n = n - c: PRINT "осталось "; n; " предм."
         WHILE n > 1
          PRINT "Ваш ход"
          DO
           INPUT "Сколько предметов Вы берете?"; y
           IF y <> 3 AND y <> 1 AND y <> 2 THEN
            PRINT "Разрешенное количество предметов 1 или 2 или 3."
           END IF
          LOOP UNTIL y = 3 OR y = 2 OR y = 1
          PRINT "Я беру"; 4 - y; "предм., ";
          n = n - 4
          PRINT "осталось "; n; "предм."
         WEND
         PRINT "Ваш ход. Вы проиграли"
```

```
  END IF
  PRINT "Если хотите сыграть еще, нажмите D (d)"
  INPUT w
LOOP UNTIL w <> "d" AND w <> "D"
END
```

3.2. Ханойская башня

Задача 3. Несколько дисков различных размеров нанизаны на один из нескольких стержней, образуя пирамидку (башню), чем выше находится диск, тем меньше его диаметр. Требуется переместить эту пирамидку на другой стержень, соблюдая следующие правила: за один раз можно перекладывать только один диск; нельзя класть диск на меньший по диаметру диск; можно пользоваться только одним резервным стержнем[15].

Начальное и конечное положения четырех дисков (пирамидки) при перекладывании с первого стержня на третий

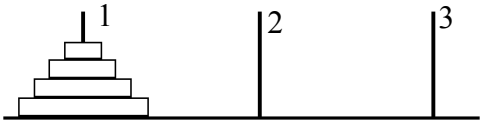


Рис. 1

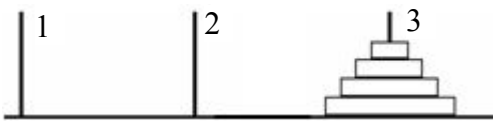


Рис. 2

Напишите алгоритм перемещения башни из *n* дисков со стержня 1 на стержень 3 при любом значении *n*:

а) Найдите метод решения задачи с применением рекурсии и метод, в котором рекурсия не используется.

б) Напишите алгоритм решения задачи в виде блок-схемы.

в) Напишите программу на языках QBasic, Ершол, Turbo Pascal.

---

[15] *Легенда о происхождении игры «Ханойская башня»*

Игру «Ханойская башня» изобрёл французский математик Люк больше ста лет назад, в 1883 году.

Где-то в непроходимых джунглях, недалеко от города Ханоя, есть монастырь бога Брамы. Когда Брама создал Мир, он воздвиг в этом монастыре три высоких алмазных стержня и на один из них возложил 64 диска, сделанных из чистого золота. Брама приказал монахам перенести эту башню на другой стержень (в соответствии с правилами). С этого времени монахи работают день и ночь. Конец Мира наступит тогда, когда все 64 диска будут перемещены, на что потребуется чуть больше 58 млрд. лет, если на перекладывание одного диска монахи будут тратить одну секунду.

Решение. Эта задача очень легко решается с использованием рекурсии, но достаточно сложно, когда рекурсия не допускается.

Алгоритм перекладывания n дисков со стержня A на стержень B

A. Рассмотрим *алгоритм1* перекладывания 2 дисков со стержня 1 на стержень 3.

Словесное описание алгоритма1 с графической иллюстрацией ходов

Исходное положение пирамидки:

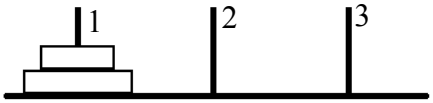


Рис. 3

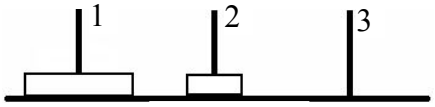


Рис. 4

1. Перенеси малый диск с исходного стержня на промежуточный.

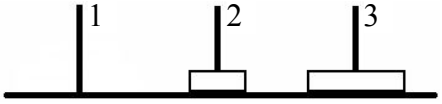


Рис. 5

2. Перенеси большой диск с исходного стержня на конечный.

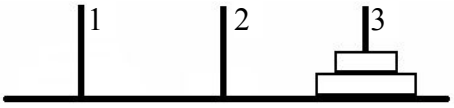


Рис. 6

3. Перенеси малый диск с промежуточного стержня на конечный. Конечное положение пирамидки.

Чтобы запись алгоритма сделать короче, обозначим ходы тремя числами: первое число означает количество перекладываемых дисков, второе – номер стержня, с которого диск снимается, третье – номер стержня, на который диск надевается.

Короткая формализованная запись алгоритма1 перекладывания двух дисков с первого стержня на третий: 1, 1 – 2; 1, 1 – 3; 1, 2 – 3.

B. Рассмотрим *алгоритм2* перекладывания трех дисков со стержня 1 на стержень 2.

Словесное описание алгоритма2 с графической иллюстрацией ходов

Исходное положение пирамидки:

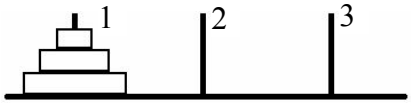


Рис. 7

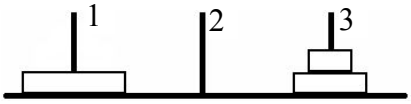


Рис. 8

1. Перенеси два верхних диска с исходного стержня на промежуточный.

2. Перенеси нижний диск с исходного стержня на конечный.

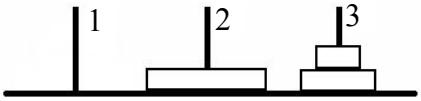


Рис. 9

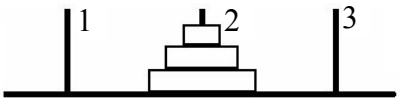


Рис. 10

3. Перенеси два диска с промежуточного стержня на конечный. Конечное положение пирамидки.

Короткая формализованная запись алгоритма2 перекладывания трех дисков с первого стержня на второй: 2, 1 – 3; 1, 1 – 2; 2, 3 – 2.

Алгоритм1 перекладывания двух дисков с одного стержня на другой известен, детализируем алгоритм2 перекладывания трех дисков с первого стержня на второй: 1, 1 – 2; 1, 1 – 3; 1, 2 – 3; 1, 1 – 2; 1, 3 – 1; 1, 3 – 2; 1, 1 – 2.

Зная алгоритм переноса трех дисков, легко написать алгоритм переноса четырех дисков и т.д.

C. Алгоритм перекладывания дисков с одного стержня на другой можно обобщить для произвольного n.

Назовем Hanoi(n,A,B) процедуру переноса пирамидки из n дисков со стержня A на стержень B.

Алгоритм3 переноса n дисков со стержня A на стержень B имеет вид:

1. Hanoi (n-1,A,C).
2. Перенеси 1 диск со стержня A на стержень B.
3. Hanoi (n-1,C,B).

Этот способ описания алгоритма3 позволяет получить изящную рекурсивную программу (смотрите программу на стр. 127).

D. Определим количество перекладываний при перемещении пирамидки из n дисков со стержня A на стержень B

Таблица количества ходов при игре в Ханойскую башню

Число дисков	Число ходов
1	1
2	3
3	7
4	15
5	31
…	…

Подсчитать количество перекладываний дисков можно двумя способами.

Первое решение	Второе решение
1=0*2+1	$1=2^1-1$
3=1*2+1	$3=2^2-1$
7=3*2+1	$7=2^3-1$
15=7*2+1	$15=2^4-1$
31=15*2+1	$31=2^5-1$
…	…
$k_n=k_{n-1}*2+1$	$k'_n=2^n-1$

k_n и k'_n – n-ые члены соответственно первой и второй последовательностей.

E. Распишем перекладывание четырех дисков со стержня A на B, используя в качестве вспомогательного диск C, используем *алгоритм3*.

Таблица шагов алгоритма3 перенесения n дисков(n=4)
со стержня A на стержень B, используя промежуточный стержень C

Позиция	Шаги алгоритма	Постепенная детализация шагов алгоритма			
	1				1, A→C
	2			2, A→B	1, A→B
	3				1, C→B
2^{n-2}	4		3, A→C	1, A→C	1, A→C
	5				1, B→A
	6			2, B→C	1, B→C
	7				1, A→C
2^{n-1}	8	4, A→B	1, A→B	1, A→B	1, A→B
	9				1, C→B
	10			2, C→A	1, C→A
	11				1, B→A
$2^{n-1}+2^{n-2}$	12		3, C→B	1, C→B	1, C→B
	13				1, A→C
	14			2, A→B	1, A→B
$2^{n}-1$	15				1, C→B

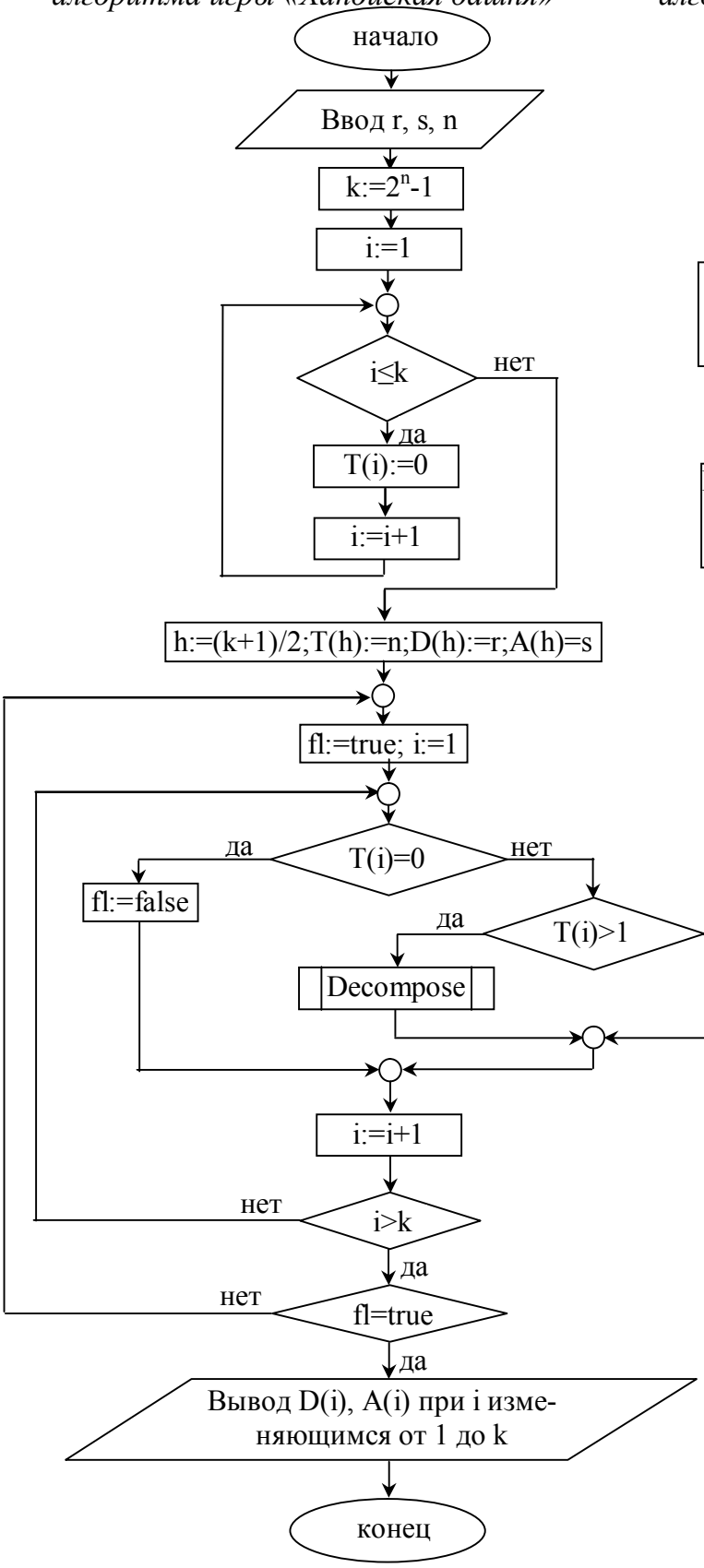
F. Построим блок-схему и напишем программу перекладывания n дисков со стержня A на стержень B, не используя рекурсию.

Необходимо совершить $2^{n}-1$ шагов и на каждом шаге расписать, с какого стержня, на какой стержень нужно переложить верхний диск. Будем расписывать перекладывания в таблице по столбцам.

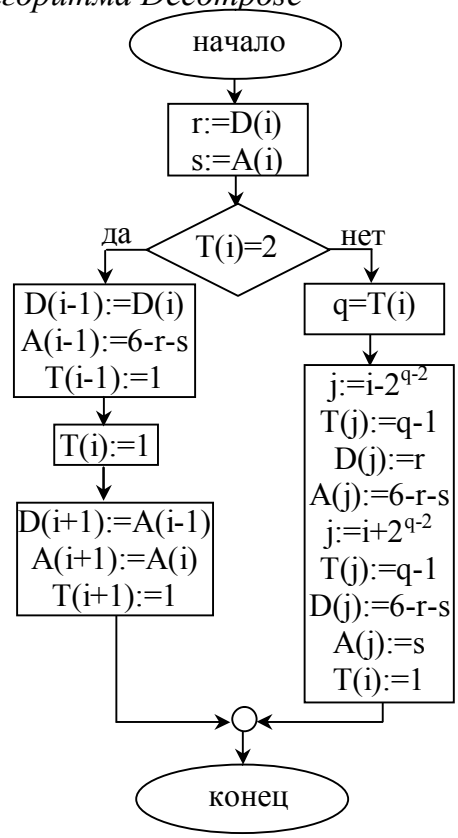
Таблица разложения на шаги перекладывания n дисков
со стержня A на стержень B

Позиция	T(i)	D(i)	A(i)	T(i)	D(i)	A(i)	…
1	0			0			…
…	…			…			…
2^{n-2}	0			n-1	A	6-A-B	…
…	…			…			…
2^{n-1}	n	A	B	1	A	B	…
…	…			…			…
$2^{n-1}+2^{n-2}$	0			n-1	6-A-B	B	…
…	…			…			…
$2^{n}-1$	0			0			…

Блок-схема основного алгоритма игры «Ханойская башня»



Блок-схема вспомогательного алгоритма Decompose



Для решения задачи возьмем 3 массива. Массив *T*, где T(*i*) – количество перекладываемых дисков в позиции *i*; *i* – номер строки таблицы; D(*i*) – указывают исходный стержень для перекладывания T(*i*) дисков в *i* строке; A(*i*) – конечный (целевой) стержень для строки *i*. Если T(*i*) = 0, то строка *i* еще не расписана. Если T(*i*)=1, то задан перенос диска со стержня, указанного D(*i*), на стержень, указанный A(*i*). Если T(*i*)>1, то задана операция Hanoi(T(*i*),D(*i*),A(*i*)), которая должна быть разложена на элементарные операции. Первоначально массив T(*i*) заполняется нулями. Разложение осуществляется то тех пор, пока все T(*i*) не получат значение 1.

Программа на языке QBasic игры Ханойская башня (без использования рекурсии)[16]

```
DECLARE SUB Decompose ()
DIM SHARED r AS INTEGER, s AS INTEGER, n AS INTEGER, k AS INTEGER
DIM SHARED i AS INTEGER, h AS INTEGER, fl AS INTEGER
INPUT "Введите количество перекладываемых дисков"; n
k = 2 ^ n - 1
DIM SHARED t(1 TO k) AS INTEGER, d(1 TO k) AS INTEGER
DIM SHARED a(1 TO k) AS INTEGER
INPUT "С какого стержня переложить (1, 2, 3)"; r
INPUT "На какой стержень переложить (1, 2, 3)"; s
FOR i = 1 TO k
  t(i) = 0
NEXT i
h = 2 ^ (n - 1)          'Начальная установка: n дисков перенести со стержня r (d(h))
t(h) = n : d(h) = r : a(h) = s   'на стержень s (a(h))
DO
  fl = 1
  FOR i = 1 TO k
    IF t(i) = 0 THEN
              fl = 0
            ELSEIF t(i) > 1 THEN
                        Decompose
    END IF
  NEXT i
LOOP UNTIL fl = 1
```

[16] Все данные алгоритма игры Ханойская башня опишем в основном алгоритме, сделаем их глобальными, и во вспомогательный алгоритм передавать не будем, т.к. в этом случае они в нем определены.

```
FOR i = 1 TO k  'Распечатка результата исполнения программы
  IF n <= 4 THEN
          PRINT USING "#####"; t(i); d(i); a(i)
         ELSE
          PRINT USING "#####"; t(i); d(i); a(i);
          PRINT " ***";
  END IF
NEXT i
END
SUB Decompose 'Разложение переноса t(i) дисков со стержня d(i) на a(i)
DIM q AS INTEGER, j AS INTEGER
'Разложение переноса m дисков со стержня r на стержень s на три переноса
'1) m-1 диск со стержня r на стержень 6-r-s
'2) 1 диск со стержня r на s
'3) m-1 диск со стерня 6-r-s на стержень s
  r = d(i): s = a(i)
  IF t(i) = 2 THEN
             t(i - 1) = 1 : d(i - 1) = d(i) : a(i - 1) = 6 - r - s : t(i) = 1
             t(i + 1) = 1 : d(i + 1) = a(i - 1) : a(i + 1) = a(i)
           ELSE
            q = t(i) : j = i - 2 ^ (q - 2) : t(j) = q - 1 : d(j) = r : a(j) = 6 - r - s
            j = i + 2 ^ (q - 2) : t(j) = q - 1 : d(j) = 6 - r - s : a(j) = s : t(i) = 1
  END IF
END SUB
```

Программа на языке Turbo Pascal игры Ханойская башня (без использования рекурсии)

```
Program Hanoi; {Ханойская башня, программа без использования рекурсии}
Const n1=10; k1=1023;
Var
  r, s: 1..3; n: 0..n1;
  k, i, h: 1..k1;
  fl: boolean;
  t: array[1..k1] of 0..n1;
  d,a: array[1..k1] of 1..3;
Function Step(x,y:integer):integer;
Var o,v:integer;
Begin
  v:=1;
```

```
  for o:=1 to y do
    v:=v*x;
  Step:=v
end;
Procedure Decompose; {Разложение переноса t[i] дисков со стержня d[i] на a[i]}
Var
  q:0..n1;
  j:1..k1;
Begin
  {Разложение переноса m дисков со стержня r на стержень s на три переноса
   1) m-1 диск со стержня r на стержень 6-r-s
   2) 1 диск со стержня r на s
   3) m-1 диск со стерня 6-r-s на стержень s}
  r:=d[i];   s:=a[i];
  If t[i]=2 then
             begin
       t[i-1]:=1; d[i-1]:=d[i]; a[i-1]:=6-r-s; t[i]:=1; t[i+1]:=1; d[i+1]:=a[i-1]; a[i+1]:=a[i]
             end
           else
             begin
               q:=t[i]; j:=i-Step(2,q-2); t[j]:=q-1; d[j]:=r; a[j]:=6-r-s; j:=i+Step(2,q-2);
               t[j]:=q-1; d[j]:=6-r-s; a[j]:=s; t[i]:=1;
             end
end;
BEGIN
  Write('Введите количество перекладываемых дисков');
  Readln(n);
  k:=Step(2,n)-1;
  Write('С какого стержня переложить (1, 2, 3)');
  Readln(r);
  Write('На какой стержень переложить (1, 2, 3)');
  Readln(s);
  For i:=1 to k do
    t[i]:=0;
  h:=Step(2,n-1);      {Начальная установка: n дисков перенести со стержня r (d[h])}
  t[h]:=n; d[h]:=r; a[h]:=s;  {на стержень s (a[h])}
  Repeat
    fl:=true;
```

```
   For i:=1 to k do
    If t[i]=0 then
              fl:=false
            else
              If t[i]>1 then Decompose
  Until fl;
  For i:=1 to k do {Распечатка результата исполнения программы}
   If n<=4 then
          Writeln(t[i]:4, d[i]:4, a[i]:4)
         else
          Write(t[i]:4, d[i]:4, a[i]:4, ' ***')
END.
```

Программа на языке Turbo Pascal игры Ханойская башня (с использованием рекурсии)

```
Program B_Hanoi; {Ханойская башня, программа с использованием рекурсии}
Uses crt;
Var a,b,c,n:integer;
Procedure Hanoi(i:integer;s,d,e:integer);
Begin
   If i=1 then
              Writeln(' ',s,'-',d)
           else
             begin
                Hanoi(i-1,s,e,d);        Hanoi(1,s,d,e);        Hanoi(i-1,e,d,s);
             end;
End;
Begin
    Write ('введите кол-во дисков n='); Readln(n);
    Writeln ('укажите начальный, конечный и промежуточный стержни:');
    Readln(a); Readln(b); Readln(c);   Hanoi(n,a,b,c);
End.
```

3.3 Игра «Жизнь»

Игра «Жизнь», придумана американским математиком Джоном Хортоном Конуэя (Конвея), сотрудником Кембриджского университета [6]. Описание игры появилось в октябрьском номере журнала

«Scientific American» в 1970 году[17]. Возникающие в процессе игры ситуации очень похожи на реальные процессы, происходящие при зарождении, развитии и гибели колонии живых организмов. По этой причине «Жизнь» можно отнести к быстро развивающейся в наши дни категории игр, имитирующих процессы, происходящие в живой природе. Эта игра необычна и красива.

Пусть «Жизнь» – многоклеточное сообщество, населяющее пустыни Флатландии. Для определенности предположим, что пустыня – прямоугольное поле, разделенное на квадратные клетки, каждая из которых может содержать организм – один элемент «жизни». Организмы умирают, рождаются, меняются поколения. Основная идея игры состоит в том, чтобы, начав с какого-нибудь простого расположения «организмов», находящихся по одному в клетке, проследить за эволюцией исходной конфигурации под действием «генетических законов» Конуэя, которые управляют рождением, гибелью, выживанием организмов.

А. *Условия*, которым должны удовлетворять правила игры «*Жизнь*». Конуэй тщательно подбирал свои правила и долго проверял их «на практике» добиваясь, чтобы они, по возможности, удовлетворяли трем условиям:

1. Не должно быть ни одной исходной конфигурации, для которой существовало бы простое доказательство возможности неограниченного роста популяции.

2. В то же время должны существовать такие начальные конфигурации, которые заведомо обладают способностью беспредельно развиваться.

[17] Из истории. Когда появилось описание игры, Конуэй предложил премию тому, кто первым докажет или опровергнет гипотезу, согласно которой не существует ни одной начальной конфигурации, способной беспредельно расти. В ноябре 1970 г. Конуэю пришлось выдать обещанную премию. Группа математиков из Массачусетского технологического института сумела построить «ружьё», стреляющее «планерами». На сороковом ходу из ружья вылетает первый «планер», через каждые 30 ходов – следующий «планер» и так до бесконечности. С появлением каждого «планера» число фишек на доске увеличивается на 5, следовательно, происходит неограниченный рост популяции.

3. Должны существовать простые начальные конфигурации, которые в течении значительного промежутка времени растут, претерпевают разнообразные изменения и заканчивают свою эволюцию одним из следующих трех способов:

а) полностью исчезают (либо из-за перенаселенности, то есть от слишком большой плотности организмов, либо из-за разреженности, то есть от одиночества организмов, образующих конфигурацию);

б) переходят в устойчивую конфигурацию и перестают изменяться вообще;

в) выходят на колебательный режим, то есть – бесконечный цикл превращений с определенным периодом.

Короче, правила должны быть такими, чтобы поведение популяции было непредсказуемым.

В. *Правила игры* «*Жизнь*». Каждую клетку окружают 8 соседних клеток (клетка имеет 8 соседей), 4 соседние клетки имеют с ней общие стороны и 4 общие вершины (клетки, расположенные рядом с ней по горизонтали, вертикали и диагоналям).

M-1, K-1	M-1, K	M-1, K+1
M, K-1	Клетка M, K	M, K+1
M+1, K-1	M+1, K	M+1, K+1

Новое поколение организмов получается из предыдущего по простым правилам:

1. Выживание**.** Каждый организм, имеющий 2 или 3 соседних организма выживает и переходит в следующее поколение.

2. Гибель**.** Каждый организм, у которого больше 3 соседей, погибает из-за перенаселения. Каждый организм, вокруг которого свободны все соседние клетки или же занята одна клетка, погибает от одиночества.

3. Рождение. Если рядом с пустой клеткой оказывается ровно 3 клетки, в которых есть жизнь (организмы), то в ней рождается новый организм.

Важно понять, что гибель и рождение всех «организмов» происходит одновременно. Вместе взятые, они образуют одно

поколение или, как мы будем говорить, один «ход» в эволюции начальной конфигурации.

Поскольку рождение и гибель «организмов» происходит одновременно, новорожденные «организмы» никак не влияют на гибель и рождение остальных организмов и поэтому, проверяя новую конфигурацию, необходимо уметь отличать их от организмов, перешедших из предыдущего поколения.

C. *Некоторые варианты моделирования игры*. Можно изображать ходы фишками двух цветов, можно рисовать ходы на бумаге либо переставлять фишки или шашки на доске. Начав игру, Вы сразу заметите, что популяция непрерывно претерпевает необычные, нередко очень красивые и всегда неожиданные изменения.

D. *Возможные эволюции исходной конфигурации колонии организмов*. В большинстве случаев исходные конфигурации либо переходят в устойчивые (их Конуэй назвал «любителями спокойной жизни») и перестают изменяться, либо навсегда переходят в колебательный режим, либо исчезают. Конфигурации, не обладавшие в начале игры симметрией, обнаруживают тенденцию к переходу в симметричные формы. Обретенные свойства симметрии в процессе дальнейшей эволюции не утрачиваются, симметрия конфигурации может лишь обогащаться.

E. *Алгоритм получения конфигураций колонии организмов в игре «Жизнь»*.

Алгоритм одного хода игры. Попробуем составить алгоритм одного «хода» игры, т.е. получения новой конфигурации из старой. Первоначально алгоритм можно записать в таком виде:

1. Занеси в клетки пустыни исходную конфигурацию жизни, т.е. в некоторые клетки помести символ, которым ты будешь изображать жизнь организма, например • или *.

2. Определи для каждой клетки пустыни, будет ли в ней жизнь в следующем поколении, в соответствии с заданными правилами получения нового поколения.

3. Построй новую конфигурацию.

Рассмотрим эти три шага подробнее.

1 шаг. Занеси в клетки пустыни исходную конфигурацию жизни**.**

Для выполнения первого шага нужно найти клетки, в которые следует занести • или ∗. Заметим, что каждую клетку пустыни можно охарактеризовать двумя числами – номерами строки и столбца. Назовем их индексами клетки. Для человека совсем не трудно найти по индексам нужные клетки и поставить в них • или ∗, т.к. он видит и умеет считать.

2 шаг. Определи для каждой клетки пустыни, будет ли в ней жизнь в следующем поколении.

Второй шаг уже не так прост, как первый; будет ли жизнь в какой-то клетке в следующем поколении, зависит от нескольких условий.

Алгоритм1 *определения «есть ли жизнь в данной клетке» по исходной конфигурации*

Чтобы определить есть ли жизнь в каждой конкретной (фиксированной) клетке поля данной конфигурации нужно выполнить следующий алгоритм1:

2.1 Подсчитай число живых соседей данной клетки, присвой это число переменной k, т.е. подсчитай, в каких соседних клетках есть • или ∗. Перейди на шаг 2.2**.**

2.2 Посмотри, есть ли в данной клетке жизнь, если нет жизни, то перейди на шаг 2.3, иначе на шаг 2.4.

2.3 Сравни число k с 3, если эти числа равны, то поставь в клетке • или ∗ для следующего поколения. Перейди на шаг 2.5.

2.4 Определи, верно ли, что $k<2$ или $k>3$, если да, то убери • или ∗ из клетки (сделай ее пустой для следующего поколения). Перейди на шаг 2.5.

2.5 Закончи алгоритм определения «есть ли жизнь в данной клетке».

3 шаг. Построй новую конфигурацию.

Алгоритм2 – заполнения каждой клетки новой «жизнью»

3.1 Если в клетке народившийся организм, зафиксируй в ней жизнь (занеси в клетку • или *) и перейди на 3.3, иначе – на 3.2.

3.2 Если в клетке погибший организм, то сделай клетку пустой и перейди на шаг 3.3, иначе оставь состояние клетки без изменения и перейди на шаг 3.3.

3.3 Закончи алгоритм заполнения каждой клетки новым состоянием, новой «жизнью».

Очевидно, что *Алгоритм1* и *Алгоритм2* выполняются столько раз, сколько клеток в пустыне, т.е. они идентичны (одинаковы) для всех клеток.

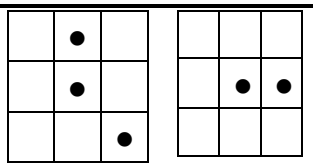
Реализация одного хода игры на языке QBasic

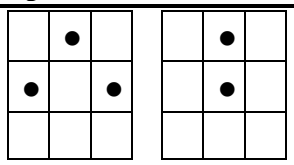
Для проведения одного хода игры следует выделить два поля. На одно наносится исходная конфигурация жизни и определяется для каждой клетки, есть ли в ней «жизнь», на другом поле строится новая конфигурация – следующее поколение. Можно проследить за эволюцией начальной конфигурации, повторяя ходы игры. Для реализации игры на компьютере в системе QBasic (Turbo Pascal) смоделируем каждое поле в структуру языка – массив A(1:*n*) и B(1:*n*) соответственно, значение элемента массива «*» означает жизнь, а значение элемента «-» означает, что в клетке пустыни жизни нет (блок-схема стр. 137-138, программа стр. 139-140)**.**

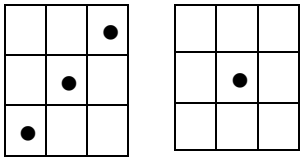
F. *Эволюция некоторых простых конфигураций колоний организмов.* Рассмотрим как изменяются некоторые простые конфигурации колоний организмов при игре.

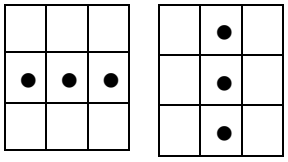
1. Один организм, а также любая пара организмов, где бы они не стояли, очевидно, погибают после первого же хода.

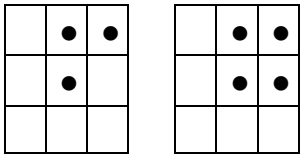
2. Триплеты – конфигурации из трех фишек. Выживает триплет лишь в том случае, если, по крайней мере, одна клетка, в которой есть организм, имеет общие стороны с двумя занятыми клетками.

a)  Погибает.

b)  Погибает.

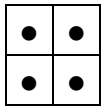
c)  Погибает.

d)  Мигалка. Период равен двум ходам.

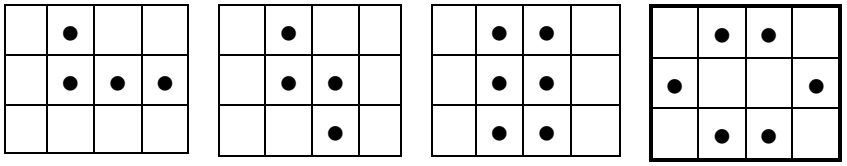
e)  Блок. Устойчивая конфигурация (любитель спокойной жизни).

3. Любой диагональный ряд организмов, каким бы длинным он не был, с каждым ходом теряет стоящие на его концах фишки и в конце концов совсем исчезает.

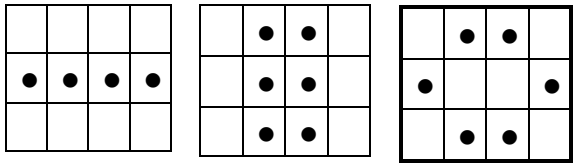
4. Тетрамино. На рисунках a-e изображена эволюция пяти тетрамино (четыре клетки, из которых состоит элемент тетрамино, связаны между собой ходом ладьи).

a) 

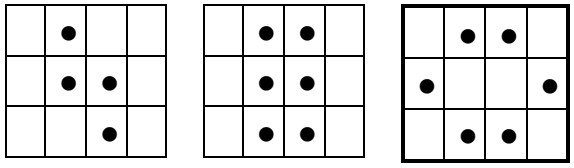
Блок – любитель спокойной жизни.

b) 

Улей – устойчивая конфигурация.

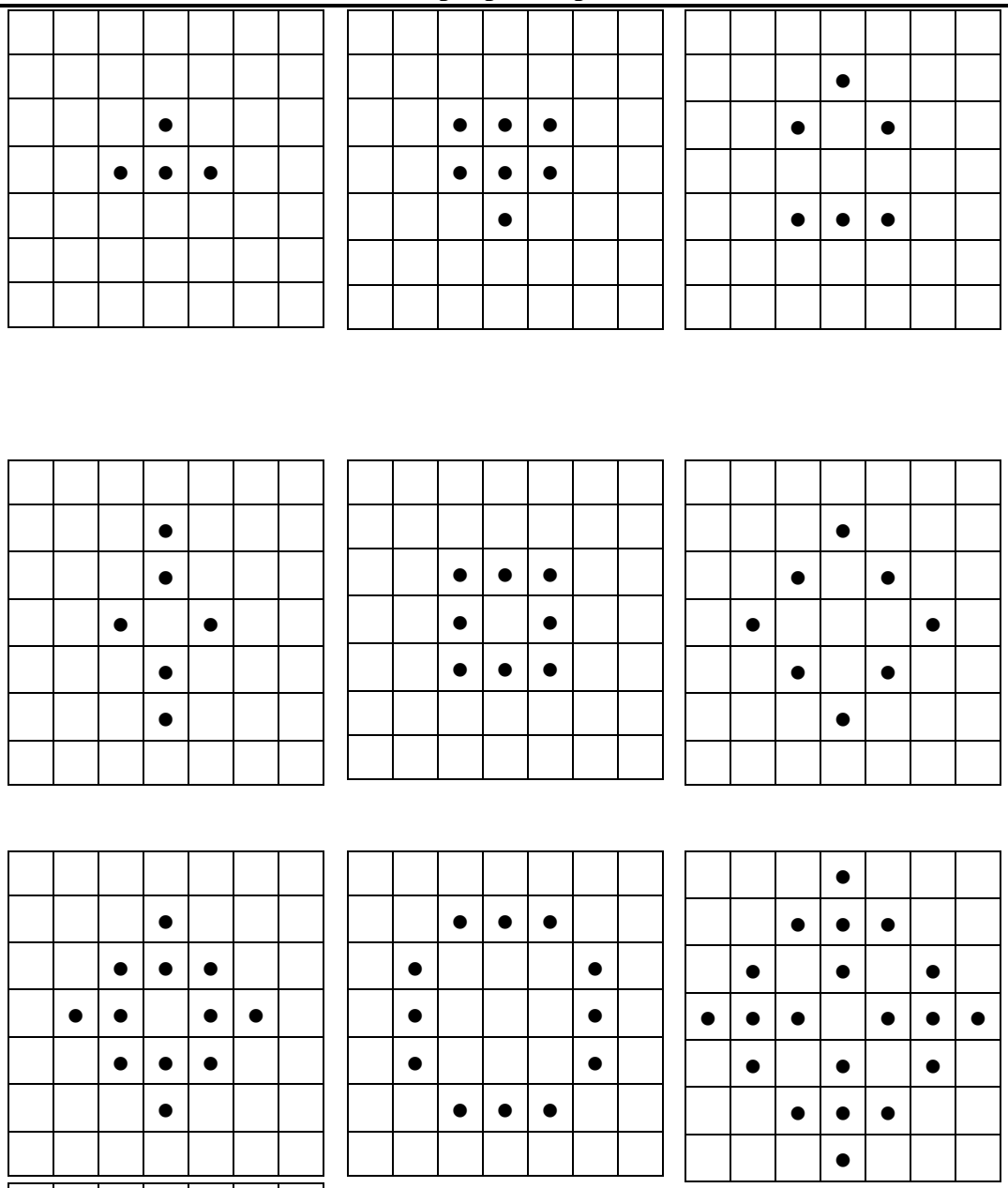
c) 

Улей

d) 

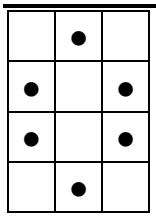
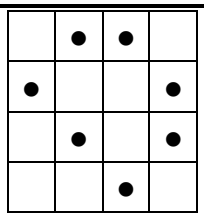
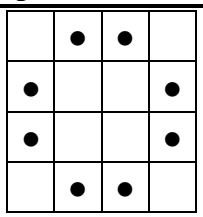
Улей

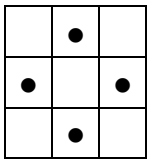
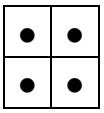
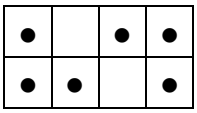
е)

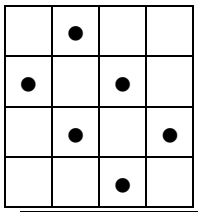
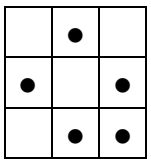
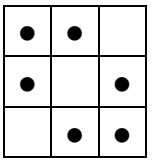


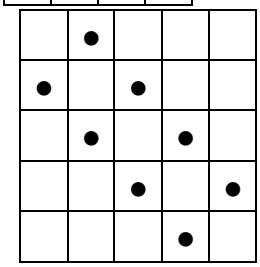
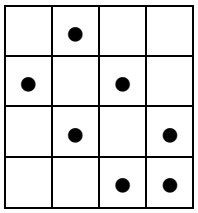
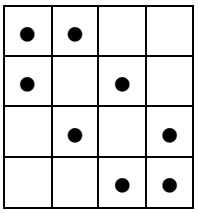
Вся конфигурация е) называется «навигационные огни».

5. Наиболее часто встречающиеся устойчивые конфигурации.

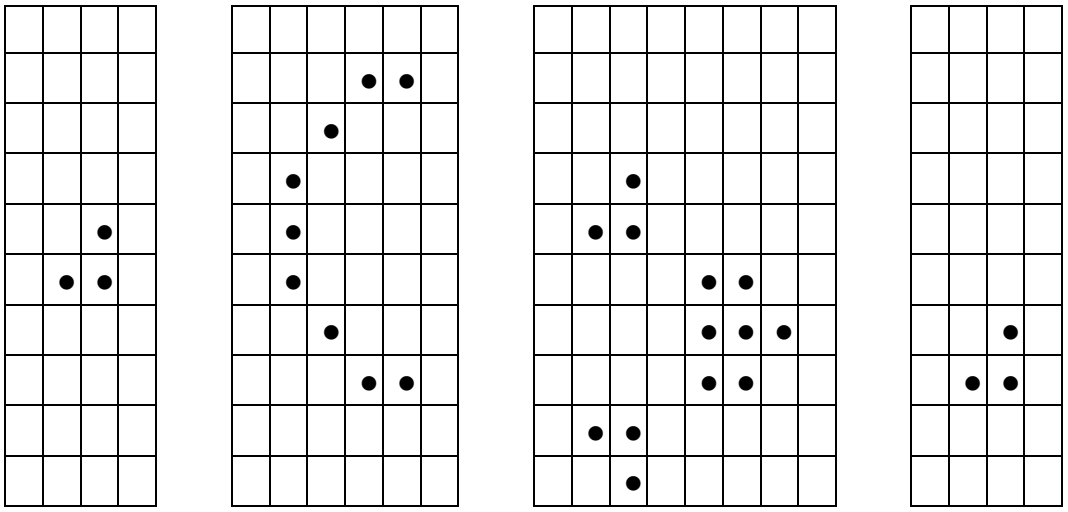

Улей

Каравай

Пруд


Ящик

Блок

Змея


Баржа

Лодка

Корабль

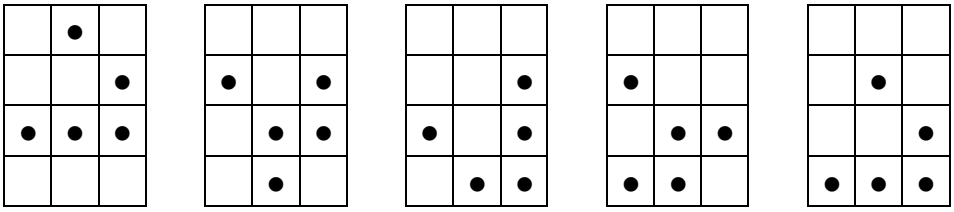

Длинная баржа

Длинная лодка

Длинный корабль

6. На рисунке изображены фрагменты конфигурации, расстояния между которыми 6 рядов, 7 рядов и 3 ряда соответственно, которая превращается в «ружье», стреляющее «глайдерами». На сороковом ходу из «ружья» вылетает первый «глайдер», через каждые 30 ходов – следующий «глайдер» и так до бесконечности. В результате чего происходит неограниченный рост популяции.



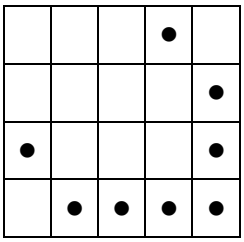
7. Перемещающиеся по полю конфигурации.

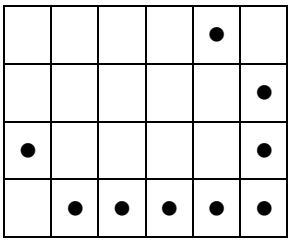
Планер (космический корабль легчайшего веса).

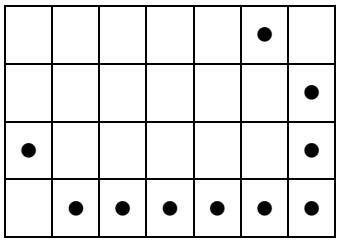


Планер переходит сам в себя после четырех ходов и при этом опускается на одну клетку вправо и на одну клетку вниз.

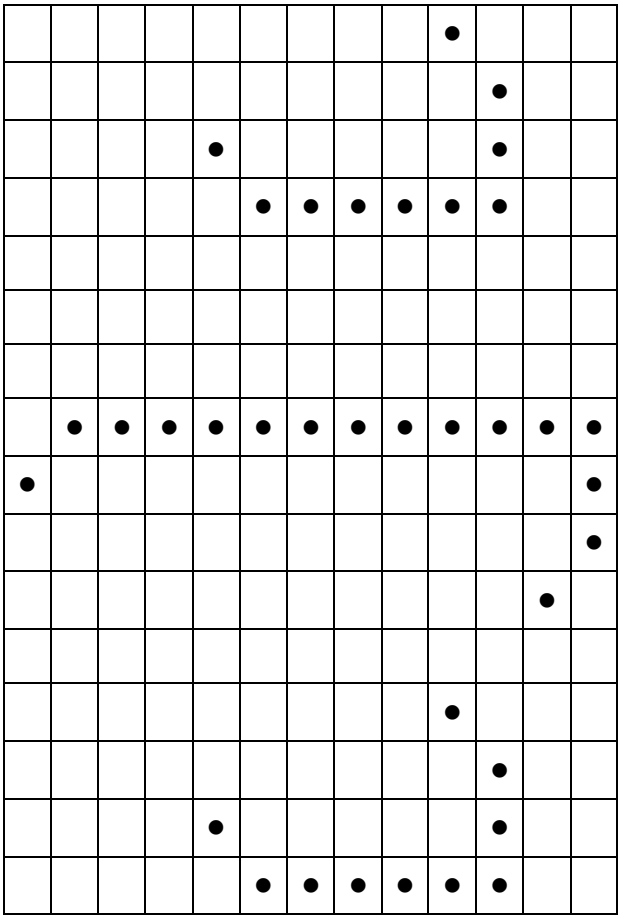
8. На рисунке изображены три «космических корабля».

a) 

b) 

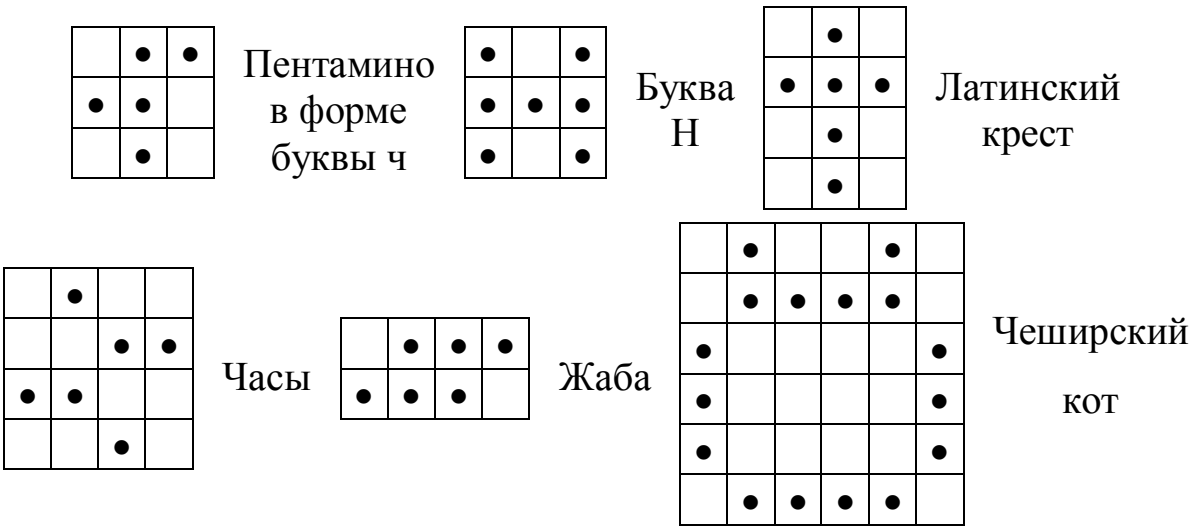
c) 

«Космические корабли» перемещаются горизонтально слева направо. В полете из них вылетают «искры», которые тут же гаснут при дальнейшем движении «кораблей». Одиночные «космические корабли» без эскорта не могут занимать в длину более шести клеток, в противном случае на поле начинают появляться различные мелкие фигуры, препятствующие движению корабля. Для более длинных «космических кораблей» необходим эскорт из двух и более «кораблей» меньших размеров.

d) 

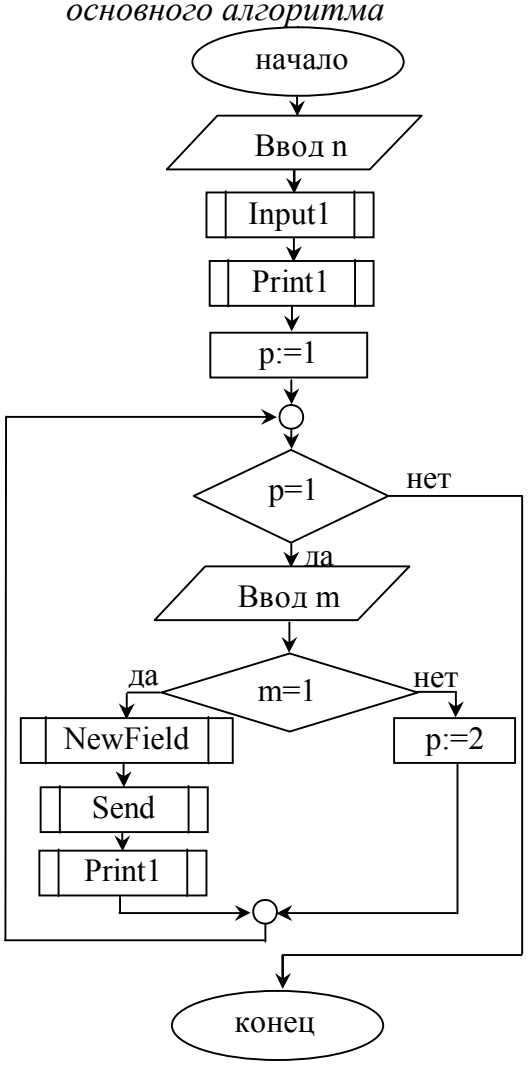
На рисунке d изображен большой «космический корабль», для которого достаточно двух эскортирующих «кораблей» меньшего размера.

9. Задание. Проследите до конца эволюцию следующих фигур:

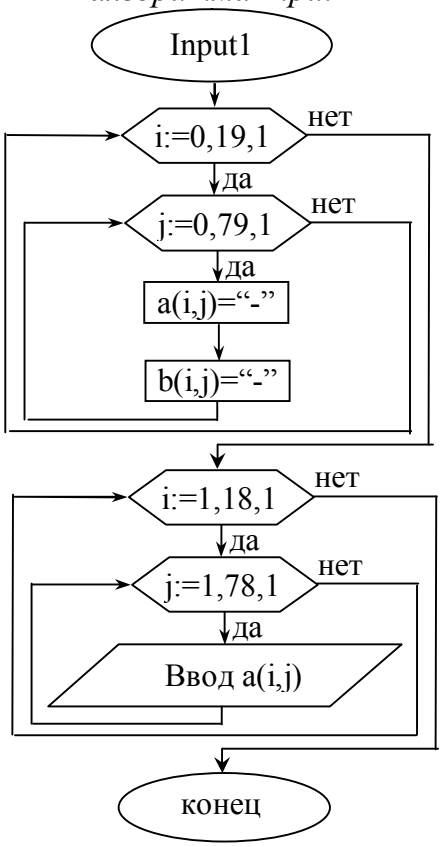


Блок-схемы алгоритма получения конфигураций колонии организмов в игре «Жизнь»

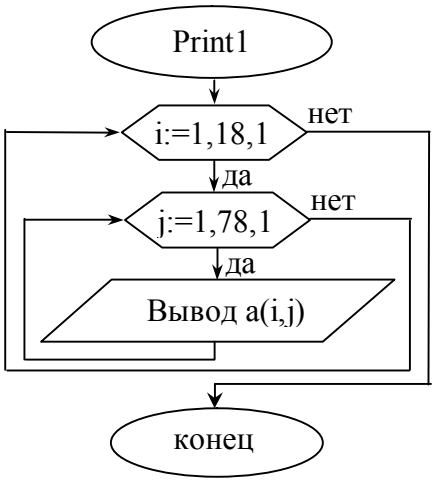
Блок-схема основного алгоритма



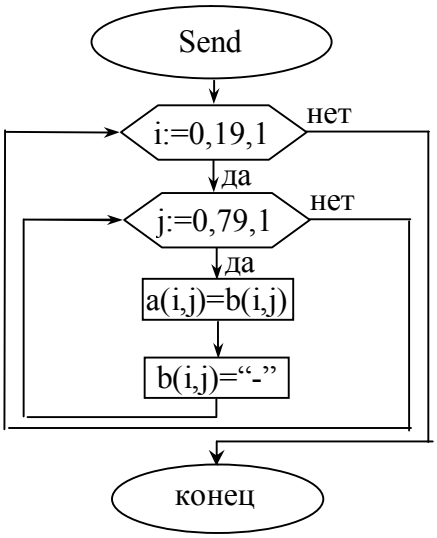
Блок-схема вспомогательного алгоритма Input1



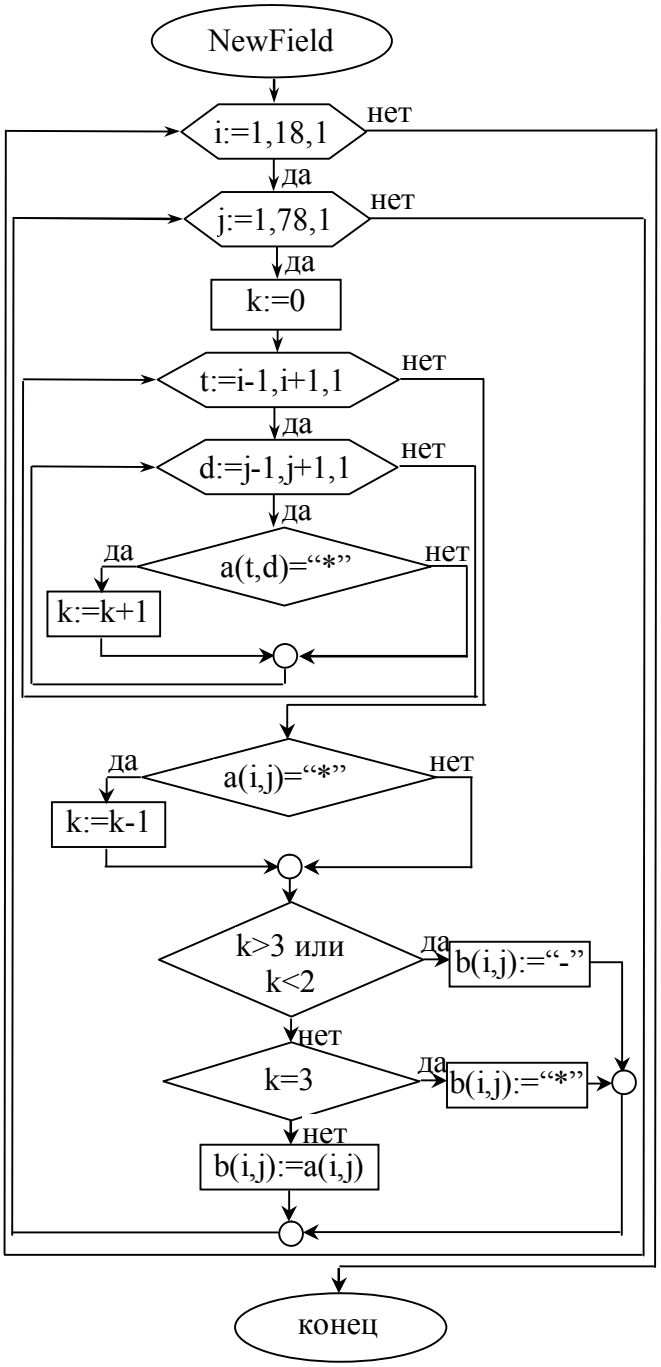
Блок-схема вспомогательного алгоритма Print1



Блок-схема вспомогательного алгоритма Send



Блок-схема вспомогательного алгоритма NewField[18]



[18] В представленной программе поле игры имеет размеры: строки – от 0 до 19, столбцы – от 0 до 79, причем крайние строки и столбцы являются границами полей и в их клетках жизни быть не может.

Программа на языке QBasic игры «Жизнь»

```
DECLARE SUB Print1 ()
DECLARE SUB NewField ()
DECLARE SUB Send ()
DECLARE SUB Input1 ()
REM Игра Жизнь
DIM p AS INTEGER, m AS INTEGER
DIM SHARED a(0 TO 24, 0 TO 79) AS STRING * 1
DIM SHARED b(0 TO 24, 0 TO 79) AS STRING * 1
CLS
Input1
Print1
p = 1
WHILE p = 1
  LOCATE 22, 5
  PRINT "Показать следующую конфигурацию? (m=1 - да, иначе - нет)";
  LOCATE 23, 5
  INPUT "Введите m"; m
  SLEEP 1
  LOCATE 22, 5
  PRINT SPC(73);
  LOCATE 23, 5
  PRINT SPC(73);
  IF m = 1 THEN
           NewField
           Send
           Print1
         ELSE
           p = 2
  END IF
WEND
END
SUB Input1
DIM i AS INTEGER, j AS INTEGER, x AS INTEGER, y AS INTEGER
DIM t AS INTEGER
'начальная инициализация массивов A и B
FOR i = 0 TO 19
  FOR j = 0 TO 79
```

```
    a(i, j) = "-" : b(i, j) = "-"
  NEXT j
NEXT i
'ввод начальной конфигурации (желательно начальную конфигурацию
t = 1                                   'популяций помещать в середину поля)
LOCATE 3, 15
PRINT "Вам предлагается ввести первоначальную конфигурацию."
LOCATE 4, 15
PRINT "Для этого необходимо ввести координаты каждой клетки,"
LOCATE 5, 23
PRINT " в которой есть жизнь."
LOCATE 6, 15
PRINT "Номер строки изменяется от 2 до 17"
LOCATE 7, 15
PRINT "Номер столбца изменяется от 2 до 77"
WHILE t = 1
  LOCATE 20, 5
  PRINT "Введите координаты клетки, в которой есть жизнь"
  LOCATE 21, 5
  INPUT "Введите номер строки"; y
  LOCATE 22, 5
  INPUT "Введите номер столбца"; x
  a(y, x) = "*"
  SLEEP 1
  LOCATE 20, 5
  PRINT SPC(73);
  LOCATE 21, 5 : PRINT SPC(73);
  LOCATE 22, 5 : PRINT SPC(73); : LOCATE 22, 5
  PRINT "Будете вводить клетки в которых есть жизнь? (t=1 - да, иначе - нет)"
  LOCATE 23, 5
  INPUT "Введите t"; t
  SLEEP 1
  LOCATE 22, 5
  PRINT SPC(73); : LOCATE 23, 5 : PRINT SPC(73);
WEND
END SUB
SUB NewField
DIM k AS INTEGER, i AS INTEGER, j AS INTEGER, t AS INTEGER
```

```
DIM d AS INTEGER
FOR i = 1 TO 18
  FOR j = 1 TO 78
    k = 0
    FOR t = i - 1 TO i + 1
      FOR d = j - 1 TO j + 1
        IF a(t, d) = "*" THEN k = k + 1
      NEXT d
    NEXT t
    IF a(i, j) = "*" THEN k = k - 1
    IF k > 3 OR k < 2 THEN
                    b(i, j) = "-"
                  ELSEIF k = 3 THEN
                              b(i, j) = "*"
                            ELSE
                              b(i, j) = a(i, j)
    END IF
  NEXT j
NEXT i
END SUB
SUB Print1
DIM i AS INTEGER, j AS INTEGER
CLS
FOR i = 1 TO 18
  FOR j = 1 TO 78
    LOCATE i, j : PRINT a(i, j)
  NEXT j
NEXT i
END SUB
SUB Send
DIM i AS INTEGER, j AS INTEGER
FOR i = 0 TO 19
  FOR j = 0 TO 79
    a(i, j) = b(i, j) : b(i, j) = "-"
  NEXT j
NEXT i
END SUB
```

Глава 3. Отдельные вопросы методики преподавания учебного материала содержательной линии «Алгоритмизация и программирование»

«Нет, и не может быть, детей, которые бы не хотели трудиться. Неумение трудиться порождает нежелание, нежелание – лень. Главное средство предупреждения этих пороков – учить воспитанников самостоятельно трудиться»

В.А. Сухомлинский

Основной целью изучения предмета «Информатика и ИКТ» в общеобразовательных учреждениях является формирование информационной культуры у обучаемых. Под информационной культурой учащихся мы понимаем уровень развития личности, развитие алгоритмического, операционного типов мышления, способности личности к системному анализу и синтезу. Эти умения и навыки имеют общекультурную, общеобразовательную ценность, они нужны в современном информационном обществе каждому человеку, независимо от его профессиональной деятельности. Дальнейшее развитие школьного предмета «Информатика и ИКТ» мы видим в усилении внимания к общеобразовательным функциям этого курса, его возможностям для решения общих задач обучения, воспитания и развития школьников. Для достижения целей изучения предмета «Информатика и ИКТ», при рассмотрении методических аспектов изучения учебного материала содержательной линии «Алгоритмизация и программирование», желательно рассматривать со студентами, наряду с общепедагогическими технологиями обучения, предметные технологии обучения, такие как проектирование алгоритмов решения задач (проектирование алгоритмов «сверху вниз» и «снизу вверх»), имитационное моделирование исполнения программ компьютером, исполнителем этих программ и др.

1. Предметные технологии формирования информационной культуры учащихся

1.1. Проектирование алгоритмов «сверху вниз» и «снизу вверх»

Технология обучения проектированию алгоритмов решения задач является управляемой системой с планируемыми результатами. Существуют различные методы поиска решения задачи. Найденное, известное решение задачи обычно излагают синтетическим методом, а чтобы найти способ решения, пользуются анализом. Синтез позволяет изложить известное решение задачи быстро и четко. Однако ученику при этом трудно понять, как было найдено решение. Анализ является средством поиска решения задачи. Если при проектировании алгоритмов систематически используется технология проектирования «сверху вниз», то у обучаемого формируются навыки поиска алгоритмов решения задач.

Одним из принципов проектирования программ является принцип декомпозиции. Декомпозиция используется для разбиения программы на модули, которые затем могут быть объединены, позволяя решить основную задачу. Создание отдельных модулей может осуществляться отдельными группами разработчиков независимо друг от друга, при этом объединенная программа будет функционировать правильно.

Процесс проектирования алгоритмов решения практических задач «*сверху вниз»* состоит в следующем: задача разбивается на подзадачи, решение которых приводит к решению основной задачи. Составляется основной алгоритм, в котором вместо алгоритмов решения подзадач записываются команды обращения к еще не написанным вспомогательным алгоритмам. В команде обращения к вспомогательному алгоритму записывается имя вспомогательного алгоритма и параметры (имена аргументов для передачи во вспомогательный алгоритм и имена результатов работы вспомогательных алгоритмов). После этого так же строятся вспомогательные алгоритмы. Процесс продолжается до тех пор, пока задача не будет сведена к набору «элементарных» задач. Под «элементарными» задачами будем понимать сложные задачи, алгоритм решения которых известен, или простые

задачи, алгоритм решения которых обучаемый легко напишет. Этот способ называется так же способом последовательного уточнения алгоритма или способом пошаговой детализации алгоритма.

Процесс проектирования алгоритма *«снизу вверх»* состоит в следующем: задача разбивается на такие подзадачи, решения которых уже известны, и решение которых приводит к решению основной задачи. Вместо известных вспомогательных алгоритмов пишутся команды обращения к ним. В команде обращения к вспомогательному алгоритму записывается имя вспомогательного алгоритма и параметры, т.е. «заголовки» известных вспомогательных алгоритмов.

Обычно при проектировании алгоритмов используются технологии проектирования алгоритма и «*сверху вниз*», и «*снизу вверх*». Рассмотренные технологии позволяют создавать программы, имеющие простую, ясную, легко модифицируемую структуру. Они позволяют создавать программы, требующие минимального времени на отладку и тестирование.

Использование технологий проектирования алгоритмов «*сверху вниз*» и «*снизу вверх*» позволяет широко применять групповые технологии обучения, например, обучаемые делятся на группы для разработки вспомогательных алгоритмов, на которые разбивается основной алгоритм. При создании электронного, классического портфолио в раздел «Рабочие материалы» учащимся надо предложить включить папку «Банк алгоритмов», куда они записывают разработанные и собранные из различных источников, разобранные алгоритмы. Количественно и качественно содержимое папки необходимо регулярно оценивать. Технологии проектирования алгоритмов «*сверху вниз*» и «*снизу вверх*» показывают, какую общеобразовательную силу могут иметь подходы, заимствованные из области алгоритмизации и программирования. В рассмотренные схемы укладывается процесс исследования в самых различных областях профессиональной деятельности.

Примеры проектирования алгоритмов с использованием технологий «сверху вниз» и «снизу вверх»

Этапы изучения учебного материала содержательной линии «Алгоритмизация и программирование»

Программа по информатике предусматривает циклический принцип изучения содержания предмета, принцип дидактической спирали. Применение этого принципа при изучении учебного материала рассматриваемой содержательной линии состоит в следующем – овладение основными понятиями содержательной линии, основными управляющими командами организации действий и организация данных проводится в два этапа. Каждая переменная, участвующая в программе, должна быть введена некоторым описанием, которое приписывает этой переменной имя и тип. Изучение типов данных является весьма непростой методической задачей. Изучение организации данных нужно проводить постепенно, а, следовательно, организацию действий и организацию данных необходимо развести по времени.

1. Пропедевтический (подготовительный) этап формирования основных понятий содержательной линии «Алгоритмизация и программирование»

На данном этапе овладение основными понятиями содержательной линии, основными управляющими командами организации действий проводится без использования величин. На этом этапе рассматриваются две технологии проектирования алгоритма «снизу вверх» и «сверху вниз». Задачи берутся из практической жизни школьников, задачи управления автоматическими устройствами (роботами). Использование исполнителей типа Робот и Чертёжник для решения поставленных на данном этапе целей позволяет длительное время обходиться без понятия величины. Для индивидуализации учения в задачах варьируется состояние обстановки на поле Робота. Бездумные исполнители позволяют сделать процесс введения величин плавным и естественным. Результаты работы исполнителей можно легко изобразить на классной доске, в тетради или в системе Кумир с использованием Пульта управления. Оказался плодотворным следующий методический приём исполнения алгоритма: на уроке один ученик исполняет роль устройства управления, а другой – роль автоматического исполнителя, изображая результаты исполнения алгоритма на доске или в системе Кумир.

2. Основной этап формирования понятий содержательной линии «Алгоритмизация и программирование».

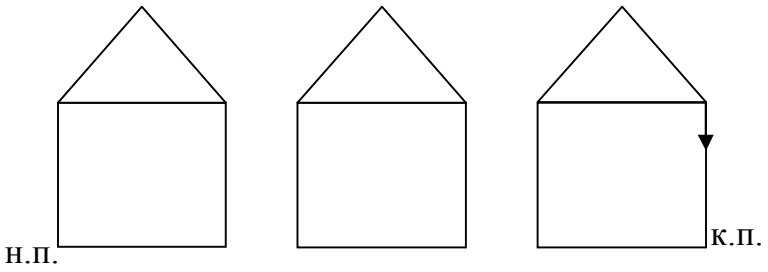
На этом этапе продолжается формирование умений и навыков использования базовых команд организации действий при решении практических задач, но уже с использованием величин. Учащиеся знакомятся с типами величин, возможностями организации величин в виде массивов, файлов, записей и т. д. Учащиеся рассматривают несколько способов ручного исполнения алгоритмов (таб-

личный, диаграммы, метод моделирования памяти), знакомятся с этапами решения задач на ЭВМ.

Главная особенность при поэтапном изучении учебного материала содержательной линии «Алгоритмизация и программирование» заключается в том, что первоначальное знакомство с основными понятиями содержательной линии проводится с опорой на *наглядные*, *житейские представления* школьников, не привлекая на первых порах ни математической символики, ни математических понятий, что заметно упрощает овладение этими понятиями. К идее привлечения *образно-наглядного мышления* школьников при введении основных понятий рассматриваемой содержательной линии независимо пришли исследователи и педагоги разных стран: Сеймур Пейперт (США), Г.А. Звенигородский (СССР), А.Г. Кушниренко (СССР) и др.

Примеры проектирования алгоритмов с использованием технологии «сверху вниз»

Пример 1. Написать алгоритм рисования улицы из трех домиков, для исполнителя Чертёжника. (Пропедевтический этап изучения учебного материала содержательной линии «Алгоритмизация и программирование»).



Решение. Для решения поставленной задачи хорошо бы уметь писать для Чертёжника алгоритмы рисования одного домика и перехода к рисованию другого домика, тогда решение поставленной задачи станет очевидным. Программа рисования улицы будет иметь вид – “алг *улица*”, табл. 1. Команды “**домик**” и “*обход*” в алгоритме “*улица*” – это команды обращения к вспомогательным, ещё не написанным алгоритмам. Эти алгоритмы необходимо написать. Как написать алгоритм рисования домика для Чертёжника? Чтобы Чертёжник нарисовал домик, хорошо бы написать алгоритмы рисования комнаты и крыши. Итак, алгоритм рисования одного домика можно написать как – “алг **домик**”, табл. 1. Команды “комната” и “*крыша*” в алгоритме “**домик**” – команды обращения к вспомогательным, ещё не написанным алгоритмам. Необходимо разработать эти алгоритмы. Алгоритмы

“комната”, “*крыша*” и “*обход*” легко написать, это элементарные алгоритмы (программы). Алгоритм “комната” имеет вид “алг комната”, табл. 1. Алгоритм “*крыша*” имеет вид “алг *крыша*”, табл. 1. Алгоритм “*обход*” имеет вид “алг *обход*”, табл. 1. Поставленная задача решена, для её решения использовалась технология проектирования алгоритма «*сверху вниз*».

алг *улица* нач поднять перо сместиться на вектор (1,1) **домик** *обход* **домик** *обход* **домик** поднять перо кон алг **домик** нач комната *крыша* кон алг *крыша* нач сместиться на вектор (2,2) сместиться на вектор (2,-2) кон	алг комната нач поднять перо сместиться на вектор (0,4) опустить перо сместиться на вектор (0,-4) сместиться на вектор (4,0) сместиться на вектор (0,4) сместиться на вектор (-4,0) кон алг *обход* нач поднять перо сместиться на вектор (0,-4) сместиться на вектор (1,0) кон

Таблица 1

Пример 2. Написать *программу упорядочения* элементов данного массива a[1:*n*] целых чисел по возрастанию, используя сортировку *простыми включениями.* (Основной этап изучения учебного материала содержательной линии «Алгоритмизация и программирование»).

Замечание: Методы сортировки обычно разделяют на две категории: сортировки массивов и сортировки файлов. Эти два класса сортировок часто называют внутренними и внешними сортировками. Массивы располагаются во «внутренней» (оперативной) памяти компьютера, для этой памяти характерен быстрый произвольный доступ. Файлы хранятся в более медленной, но более вместительной «внешней» памяти, т. е. на запоминающих устройствах.

Известные алгоритмы сортировок данных, расположенных в оперативной памяти, чрезвычайно разнообразны. Их анализ очень полезен с точки зрения обучения, так как в них используются практически все универсальные приемы конструирования алгоритмов любой сложности. По словам Н. Вирта, «создается впечатление, что можно построить целый курс программирования, выбирая примеры только из задач сортировки». Интересен этот класс алгоритмов еще и тем, что на нем наглядно демонстрируются богатейшие возможности программирования: насколько разными путями можно прийти к одной цели – получению упорядоченного массива. На множестве алгоритмов сортировки становится понятной необходимость оценки качества алгоритма, критериями которого являются экономии памяти и увеличение быстродействия. Под сортировкой обычно понимают процесс перестановки объектов данного множества в определенном порядке. Цель сортировок – облегчить последующий поиск элементов в отсортированном массиве. Упорядоченные объекты содержатся в телефонных книгах, в ведомостях подоходных налогов, в оглавлениях, в библиотеках, в словарях, на складах, да и почти всюду, где их нужно разыскивать.

Основное требование к методам сортировки массивов – экономное использование памяти. Это означает, что *переупорядочение элементов нужно выполнять в этом же массиве*. Классификацию алгоритмов сортировки проводят в соответствии с их эффективностью. Удобная мера эффективности получается при подсчете числа C – необходимых сравнений элементов и числа M – числа пересылок элементов. Эти числа определяются некоторыми функциями от n – числа сортируемых элементов. Хотя хорошие (*быстрые*) алгоритмы сортировки требуют порядка $n \cdot \log n$ сравнений, в школьном предмете информатики желательно рассмотреть несколько несложных и очевидных способов сортировки, называемых *простыми* сортировками, которые требуют порядка n^2 сравнений элементов. Необходимость рассмотрения простых методов сортировок массивов, прежде чем перейти к более быстрым сортировкам, обусловлена следующими важными причинами:

1. Простые методы особенно хорошо подходят для разъяснения принципов работы сортировок. Программы, основанные на этих методах, легки для понимания и компактны в написании.

2. Хотя быстрые методы сортировок требуют меньшего числа операций, но при достаточно малых n простые методы работают быстрее.

Простые сортировки массивов можно разбить на три основных класса, в зависимости от лежащего в их основе приема сортировки:

1. Сортировки простыми *включениями.*
2. Сортировки простым *выбором.*
3. Сортировки простым *обменом.*

Решение. Механизм сортировки простыми включениями можно описать так. Элементы массива условно разделим на две последовательности: a[1], a[2], … , a[i-1] – последовательность элементов уже отсортированную и не отсортированную последовательность a[i], a[i+1], … , a[n]. Чтобы отсортировать элементы второй последовательности поступают следующим образом, берут i-ый элемент этой последовательности и вставляют его на нужное место в отсортированную первую последовательность, i изменяется от 2 до n. Вставку элемента a[i] на нужное место в отсортированную первую последовательность производят тремя шагами.

1. Найди k – позицию (место) для элемента массива a[i] в упорядоченной последовательности элементов.

2. Сдвинь элементы a[k], a[k+1], … , a[i-1] вправо на одну позицию.

3. Поставь элемент a[i] на k место в отсортированной последовательности элементов.

Эту последовательность шагов необходимо повторить n-1 раз при i, изменяющимся от 2 до n. В начале цикла необходимо предусмотреть сохранение a[i], h:=a[i].

Алгоритмы, реализующие первый и второй шаги, оформим как вспомогательные, в основном алгоритме вместо команд алгоритмического языка, описывающих эти шаги, запишем команды обращения к

вспомогательным алгоритмам. Аргументами первого вспомогательного алгоритма «алг *место …*» являются: цел *n,i,* цел таб a[1*:n*]. Результатами исполнения алгоритма являются: цел таб a[1*:n*] и цел *k* – место a[*i*] в упорядоченной последовательности целых чисел a[1], a[2], …, a[*i*-1]. Аргументами второго вспомогательного алгоритма «алг *сдвиг вправо …*» являются: цел *k,n,i,* цел таб a[1*:n*]. Результатом исполнения алгоритма являются: цел таб a[1*:n*]. Основной алгоритм – «алг *сортировка включениями …*» приведён в таблице 2. Данные *n, i, k*, массив *a* можно описать перед первым алгоритмом как глобальные объекты.

1. *Детализируем вспомогательный алгоритм* «алг *место …*». Как найти *k*, номер a[*i*] в упорядоченной последовательности целых чисел a[1], a[2], …, a[*i*-1].

1 способ. Будем сравнивать a[*i*] по порядку со всеми элементами первой упорядоченной последовательности справа налево, начиная с элемента a[*i*-1]. Сравнение будем производить пока a[*j*]>a[*i*], где *j* изменяется от *i*-1 до 1, следовательно, сравнение нужно проводить в цикле. Как только найдётся элемент a[*j*]<=a[*i*], то и номер *k* будет найден, *k* = *j*+1. Из цикла в этом случае необходимо выйти. Возможен случай, когда все a[*j*]>a[*i*], тогда *k* = 1, при написании алгоритма эту ситуацию надо запрограммировать. Перед циклом сравнений проведём инициализацию цикла – переменной *k* присвоим значение 1, а *j* присвоим значение *i*-1: *k*:=1 и *j*:=*i*-1.

Условие, при котором цикл будет выполняться, можно записать так: пока *k*=1 и *j*>=1. Детализированный вспомогательный алгоритм «алг *место …*», реализующий эти рассуждения, дан в табл. 2.

Замечание: При программировании поиска места *k*, когда элемент a[*i*] нужно ставить на первое место, можно воспользоваться методом «барьера», названным так Н. Виртом. Расширим диапазон индексов исходного массива a[**0**:*n*], и перед циклом сравнений, при поиске места для a[*i*] в упорядоченной последовательности, запишем команду a[0]:=a[*i*]. Сравнение в этом случае нужно производить пока a[*j*]>a[*i*], где *j* изменяется от *i*-1 до **0**.

```
алг сортировка включениями (арг цел n)
нач цел таб a[1:n], цел i,k,h
    нц для i от 1 до n
        вывод "a[",i,"]="
         ввод a[i]
    кц
    i:=1
    вывод_элементов массива(i,n,a)
    нц для i от 2 до n
        h:=a[i]
        место (n,i,a,k)
        сдвиг вправо (k,i,n,a)
        a[k]:=h
        вывод_элементов массива(i,n,a)
    кц
кон
алг сдвиг вправо (арг цел k,i,n, арг рез
цел таб a[1:n])
нач цел j
    нц для j от i до k+1 шаг -1
        a[j] :=a[j-1]
    кц
кон
```

```
алг место (арг цел n,i, арг рез цел таб
   a[1:n], рез цел k)
нач цел j
    k:=1; j:=i-1
    нц пока k=1 и j>=1
        если a[j] <=a[i]
            то
                k:=j+1
        все
        j:=j-1
    кц
кон
алг вывод_элементов массива (арг цел
    i,n, арг цел таб a[1:n])
нач цел j
    вывод нс
    нц для j от 1 до n
        вывод a[j]," "
        если i=j
            то
                вывод "       "
        все
    кц
кон
```

Таблица 2

2 способ. Первая последовательность элементов массива a[1:*n*] – *a*[1], *a*[2], … , *a*[*i*-1] – отсортированная последовательность элементов. Следовательно, поиск места элемента a[*i*] необходимо проводить в отсортированной последовательности элементов. В этом случае для поиска можно использовать другой алгоритм. Сравним a[*i*] со средним элементом отсортированной последовательности. Результат сравнения позволит определить либо место элемента a[*i*], либо в какой половине последовательности следует искать позицию этого элемента. Далее, если позиция не определена, процедуру сравнения со средним элементом выбранной подпоследовательности нужно повторить. Позиция будет определена после проведения не более чем log2 i сравнений. Такой алгоритм поиска элемента в отсортированном массиве называется «*бинарным поиском*» или «*логарифмическим поиском*», а сортировка с использованием бинарного поиска места элемента a[*i*] в отсортированной последовательности называется *сортиров-*

кой бинарными включениями. Напишем вспомогательный алгоритм «алг *место1…»,* табл. 3.

```
Алг место1 (арг цел n,i, арг рез цел
таб a[1:n],рез цел k)
нач цел x,y,c
   x:=1; y:=i-1
   нц пока x<=y
      c:=div(x+y,2)
      если a[i] <a[c]
         то y:=c-1
         иначе x:=c+1
      все
   кц
   k:=x
кон
```

Таблица 3

Обозначим x номер наименьшего элемента упорядоченной последовательности, x:=1, y – номер наибольшего элемента этой последовательности, y:= i-1. Так же будем обозначать соответствующие номера выделяемых её частей (подпоследовательностей). Место включения найдено, если a[x]<=a[i]<a[y]. Итак, найдём номер c равный div(x+y, 2). Сравним элемент a[c] с элементом a[i], если значение элемента a[i]<a[c], то y:=c-1, иначе x:=c+1. Интервал поиска в конце должен быть равен 1 и k:=x. Значение k, это место a[i] в упорядоченной последовательности.

2. *Детализируем вспомогательный алгоритм* «алг *сдвиг вправо…*».

Для сдвига элементов упорядоченной последовательности a[k], a[k+1], … , a[i-1] вправо на одну позицию нужно последовательно по одному сдвигать элементы, начиная с конца на одну позицию. Детализированный вспомогательный алгоритм «алг *сдвиг вправо…*», реализующий этот шаг дан в таблице 2.

3. Алгоритм, реализующий третий шаг, элементарный алгоритм.

Для организации вывода элементов массива после выполнения команд тела основного цикла воспользуемся вспомогательным алгоритмом «алг *вывод_элементов массива ...*». Алгоритм «алг *вывод_элементов массива ...*» дан в таблице 2.

Программа сортировки простыми включениями на языке Turbo Pascal

```
program Sort_insertion;
{блок описаний основной программы}
-----------------------------------------------
const n=5;
```

```
procedure place (n, i: byte; b:
  tmass;Var k:byte);
var j:byte;
begin
```

```
type tmass=array [1..n] of byte;
var a:tmass; i,k,h:byte;
procedure place (n, i: byte; b: tmass; Var k: byte);
begin … end;
procedure displacementright (k, i, n: byte; Var b: tmass);
begin … end;
procedure print (i,n:byte; b:tmass);
begin … end;
-------------------------------------------------
begin {блок операторов основной программы}
   for i:=1 to n do
      begin
         write ('a[',i,']= '); readln(a[i]);
      end;
   i:=0;  print (i,n,a);
   i:=1;  print (i,n,a);
   for i:=2 to n do
      begin
         h:=a[i]; place (n,i,a,k);
         displacementright (k,i,n,a);
         a[k]:=h; print (i,n,a);
      end;
end.{конец основной программы}
```

```
  k:=1; j:=i-1;
  while (k=1) and (j>=1) do
     begin
        if b[j] <=b[i] then k:=j+1;
        j:=j-1
     end;
end;
-------------------------------------------
procedure displacementright (k, i, n:
   byte; Var b:tmass);
var j:byte;
begin
  for j:=i downto k+1 do
  b[j]:=b[j-1];
end;
-------------------------------
procedure print (i,n:byte; b:tmass);
var j:byte;
begin
  writeln;
  for j:=1 to n do
     begin
        write(a[j],' ');
        if i=j then write ('      ');
     end;
end;
```

Замечание: Программа на VBA для Microsoft Excel, имитирующая механизм исполнения сортировки методом простых включений, представлена в приложении 1.

Примеры проектирования программ (алгоритмов) с использованием технологии «снизу вверх»

Примеры приведены при решении задач 33, стр. 63; 35, стр. 64; при рассмотрении сортировки выбором, стр. 175.

Задания к данному параграфу

1. При разработке программ в главах 1-2 данной работы использовалось проектирование алгоритмов «сверху вниз» и «снизу вверх». Отберите по две задачи на использование этих технологий, проанализируйте их решения. Запишите условия и решения в тетради со своими комментариями, возможно, на другом языке.

2. Сформулируйте *свою* задачу, для решения которой Вы при-

менили бы изучаемую технологию проектирования алгоритмов:

- «сверху - вниз»;
- «снизу - вверх»;
- «сверху вниз» и «снизу вверх».

Определите последовательность решения задач, которые являются подзадачами основной задачи, и решение которых приводит к её решению. Разработайте содержание беседы, раскрывающей необходимость изучения выбранной Вами технологии проектирования алгоритмов (программ), её сущность и преимущества. Оформите решение рассматриваемой задачи.

1.2. Имитационное моделирование исполнения программ компьютером

Как можно использовать технологию имитационного моделирования исполнения программы компьютером при формировании информационной культуры учащихся, изучая учебный материал содержательной линии «Алгоритмизация и программирование»? *В основе этой технологии лежит имитационное или имитационно-игровое моделирование, т.е. воспроизведение в условиях обучения с той или иной мерой адекватности процессов, происходящих в реальной системе.*

Цель использования технологии имитационного моделирования на данном этапе изучения информатики: развить представление учащихся о сущности формального исполнения алгоритмов, закрепить в сознании учащихся понимание принципа пошагового формального исполнение программы компьютером по его записи, формирование приёмов умственной деятельности и умственного развития учащихся. *Сущность* способа: задаются фиксированные значения аргументов, и программа исполняется с учетом указаний, предписываемых её командами, причем значения всех величин, получаемых в результате исполнения команды, фиксируются.

Мы предлагаем рассмотреть два вида имитационного моделирования в данной содержательной линии: ручное моделирование исполнения программы компьютером и моделирование исполнения программы компьютером с использованием программных средств.

В *ручном моделировании* исполнения программы, в зависимости от способа фиксирования значений величин, получаемых в результате исполнения команды, мы вычленяем два вида ручного моделирования, каждый из которых, по нашему мнению, заслуживает применения в процессе изучения информатики:

a. *моделирование памяти компьютера* [13];
b. *моделирование с использованием наглядных протоколов*.

1.2.1. Моделирование памяти компьютера

Процесс моделирования с использованием имитации записи компьютером в оперативной памяти значений величин, используемых в программе, после исполнения каждого шага программы, состоит в следующем: память компьютера представляется в виде классной доски или листа бумаги, на которых можно записывать информацию, читать, стирать, записывать заново. *Учащимся необходимо напомнить, что значение каждой величины в компьютере хранится в отдельной ячейке, и любая величина сохраняет своё значение, пока ей не будет присвоено новое значение.* Место, отводимое в памяти компьютера под значения величин, используемых в программе, изображается в виде прямоугольника, сверху пишут, под значения величин какого алгоритма отводится эта область. Место, отводимое в памяти компьютера для каждой величины, также изображается в виде прямоугольника. Тип и имя величин указывается сверху выделенных прямоугольников, а значения величин, если они определены, записываются внутри прямоугольника. Если значение величины не определено, то в прямоугольнике ничего писать не надо. Если при выполнении команды программы меняется значение какой-то величины алгоритма, то в прямоугольник, соответствующий этой величине, заносится новое значение. Последовательно выполняемые команды записываются между рисунками, вместо имен переменных в выражениях указываются их значения, за проверяемым условием в скобках пишется значение этого условия (истинно/да или ложно/нет).

Пример 3. Напишите программу вычисления площади треугольника по трём сторонам, если заведомо известно, что треугольник су-

ществует. Исполните составленную программу, используя *метод моделирования памяти*, для $a = 3$, $b = 4$, $c = 5$.

Решение

```
алг площадь_треугольника (арг вещ a, b, c, рез вещ s) | 1 – ый способ
    дано | длины трех сторон треугольника: a, b, c
    надо |вычислить площадь s треугольника
нач вещ p
| p:=(a+b+c)/2; s:=sqrt(p*(p-a)*(p-b)*(p-c))
кон
алг площадь_треугольника  | 2 – способ
    дано | длины трех сторон треугольника: a, b, c
    надо | вычислить площадь s треугольника
нач вещ a, b, c, s, p
| вывод “Введите длины трех сторон a, b, c”; ввод a, b, c
| p:=(a+b+c)/2; s:=sqrt(p*(p-a)*(p-b)*(p-c))
| вывод нс, “Площадь равна”, s
кон
```

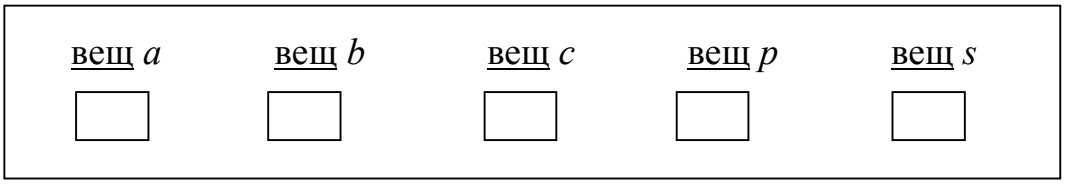
Исполнение алгоритма *«площадь_треугольника»*

Изобразим на доске или в тетради прямоугольник, сверху напишем, под значения величин какого алгоритма отводится эта область.

Место, отводимое в памяти компьютера для каждой величины, также удобно изображать в виде прямоугольника. Тип и имя величин будем указывать сверху выделенных прямоугольников, а значения величин (если они определены) записывать внутри прямоугольника.

1 шаг. Отводится место для исполняемой программы и величин, используемых в программе

алг *площадь треугольника*



2 шаг. Программа запрашивает значения a, b, c. Пользователь вводит эти значения с клавиатуры. Компьютер заносит эти значения в память. Пусть введены: $a = 3$, $b = 4$, $c = 5$.

алг *площадь треугольника*

вещ *a*	вещ *b*	вещ *c*	вещ *p*	вещ *s*
3	4	5		

3 шаг. Компьютер вычисляет значение p и заносит в память: $p = (3+4+5)/2 = 6$.

алг *площадь треугольника*

вещ *a*	вещ *b*	вещ *c*	вещ *p*	вещ *s*
3	4	5	6	

4 шаг. Компьютер вычисляет значение s и заносит его в память: $s = \sqrt{6 \cdot (6-3) \cdot (6-4) \cdot (6-5)} = \sqrt{36} = 6$.

алг площадь треугольника

вещ *a*	вещ *b*	вещ *c*	вещ *p*	вещ *s*
3	4	5	6	6

5 шаг. Компьютер выводит значение площади s на экран.

Пример 4. Напишите программу обмена значениями двух переменных m и n целого типа: переменная m должна принять значение переменной n, а переменная n – значение переменной m. Исполните составленную программу, используя *метод моделирования памяти*, для $m = 25$, $n = -10$.

Решение: Чтобы найти способ решения поставленной задачи, рассмотрим такую «*жизненную*» ситуацию. Имеется два сосуда с жидкостью, в стакане находится молоко, а в чашке – кофе с молоком. Требуется обменять содержимое этих сосудов так, чтобы в стакане было кофе с молоком, а в чашке – молоко. Возникает необходимость иметь ещё один пустой сосуд, например, бокал, для того чтобы выполнить следующие действия: перелить молоко из стакана в бокал, перелить кофе с молоком из чашки в стакан, перелить молоко из бо-

кала в чашку. Все сосуды вмещают одинаковое количество жидкости. Житейская задача решена.

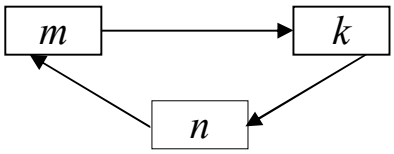
Решим *основную* задачу. Чтобы обменять значениями две переменные *m* и *n* целого типа, можно использовать ещё одну переменную целого типа, например, переменную *k*. Запрограммируем выполнение следующих действий: переменной *k* присвоим значение переменной *m, k*:=*m*, переменной *m* присвоим значение переменной *n, m*:=*n*, переменной *n* присвоим значение переменной *k, n*:=*k*. Эти действия реализованы в алгоритме «алг *обмен значениями».*

алг *обмен значениями*

дано | значения переменных *m* и *n*

надо | обменять значениями переменные *m* и *n*

```
нач цел m, n, k
    вывод "Введите значения m и n"
    ввод m, n
    k:=m; m:=n; n:=k
    вывод нс, "m=", m, "n=", n
кон
```



Исполнение алгоритма «*обмен значениями»*

1 шаг. Отводится место для исполняемой программы и величин, используемых в программе.

алг *обмен значениями*

цел *m*	цел *n*	цел *k*

2 шаг. Программа запрашивает значения *m, n*. Пользователь вводит эти значения с клавиатуры. Компьютер заносит эти значения в память. Пусть введены: *m* = 25, *n* = -10.

алг *обмен значениями*

цел *m*	цел *n*	цел *k*
25	-10	

3 шаг. Компьютер переменной *k* присваивает значение переменной *m*: *k* = 25.

алг *обмен значениями*

цел *m*	цел *n*	цел *k*
25	-10	25

4 шаг. Компьютер переменной *m* присваивает значение переменной *n*, *m* = -10.

алг *обмен значениями*

цел *m*	цел *n*	цел *k*
-10	-10	25

алг *обмен значениями*

цел *m*	цел *n*	цел
-10	25	25

5 шаг. Компьютер переменной *n* присваивает значение переменной *k*, *n* = 25.

6 шаг. Компьютер выводит значения переменных *m* и *n* на экран: *m* = -10, *n* = 25.

Пример 5. Напишите программу БИТ, нахождения числа *bid* – большего числа из трёх вещественных чисел, заданных с клавиатуры. Исполните составленную программу, используя *метод моделирования памяти*, для *m* = -5.34, *n* = -10.5, *k* = -3.3.

Решение: Чтобы выбрать большее число из трёх заданных чисел *m, n*, *k*, можно выбрать сначала большее число из двух чисел *m* и *n*, выбранное большее значение присвоить переменной *bid*. Затем выбрать большее число из чисел *bid* и *k*. Чтобы выбрать большее число из чисел *m* и *n,* необходимо сравнить эти числа, *m*>*n* (?). Если высказывание *m*>*n* – истинное высказывание, то переменной *bid* нужно присвоить значение *m*, иначе значение *n*. Далее нужно сравнить значения переменной *bid* и переменной *k*, *bid*<*k* (?). Если высказывание *bid*<*k* – истинное высказывание, то переменной *bid* присваивается значение переменной *k*, иначе значение переменной *bid* больше значения *k* или ему равно, и никаких действий производить не нужно. Эти действия реализованы в алгоритме «алг *БИТ»*.

```
алг БИТ
    дано |значения переменных m , n, k
    надо |переменной bid присвоить
         |большее значение из значений
         |переменных m, n, k
нач цел m, n, k, bid
| вывод "Введите значения m, n, k"
| ввод m, n, k
| если m>n
| | то bid:=m
| | иначе bid:=n
| все
| если bid<k
| | то bid:=k
| все
| вывод нс, "bid =", bid
кон
```

Исполнение алгоритма «БИТ»

1 шаг. Отводится место для исполняемой программы и величин, используемых в программе.

алг *БИТ*

вещ *m*	вещ *n*	вещ *k*	вещ *bid*

2 шаг. Программа запрашивает значения *m*, *n*, *k*. Пользователь вводит эти значения с клавиатуры. Компьютер заносит их в память. Пусть введены: *m* = -5.34, *n* = -10.5, *k* = -3.3.

алг *БИТ*

вещ *m*	вещ *n*	вещ *k*	вещ *bid*
-5.34	-10.5	-3.3	

3 шаг. Компьютер проверяет истинность высказывания *m*>*n* (?), то есть -5.34>-10.5. Это высказывание истинно, выполнятся команды, следующие за служебным словом «то».

4 шаг. Компьютер переменной *bid* присваивает значение переменной *m*, *bid*:=*m*, *bid* = -5.34. Значение *bid* записывается в память компьютера.

алг *БИТ*

вещ *m*	вещ *n*	вещ *k*	вещ *bid*
-5.34	-10.5	-3.3	-5.34

5 шаг. Компьютер проверяет истинность высказывания *bid*<*k* (?), то есть -5.34<-3.3. Это высказывание истинно, выполнятся команды, следующие за служебным словом «то».

6 шаг. Компьютер переменной *bid* присваивает значение переменной *k*, *bid*:=*k*, *bid* = -3.3 Значение *bid* записывается в память компьютера.

алг *БИТ*

вещ *m*	вещ *n*	вещ *k*	вещ *bid*
-5.34	-10.5	-3.3	-3.3

7 шаг. Компьютер выводит значение переменной *bid*, равное -3.3, на экран.

Задания к данному параграфу

Отберите по три задачи, рассмотренных в главе один. Оформите решение данных задач на одном из языков программирования. Исполните составленные программы, используя *метод моделирования памяти*, для подобранных вами тестов.

1.2.2. Моделирование с использованием наглядных протоколов

Процесс *имитационного моделирования* исполнения компьютером программ *с использованием наглядных протоколов* состоит в сле-

дующем: запись протокола исполнения программы осуществляется справа от записи программы, поэтому ученики должны оставить не менее половины страницы справа свободной. Проверка условий в составных командах и результаты выполнения команд присваивания необходимо отмечать в тех строках, где записаны соответствующие команды. Обход команды и выход из алгоритма надо отмечать стрелками для наглядности. В соответствующих строках протокола надо писать: $d1 = 0$, $x1 = 1$, так как это констатация факта, результат исполнения команд, а не команда на исполнение. В строках проверки условия вместо имён переменных записываются их значения, за проверяемым условием в скобках пишется значение этого условия (истинно/да или ложно/нет), например, проверяется условие $D>0$, при исполнении в этой строке пишут 1>0? (да), вместо D подставляется его значение. При исполнении циклов в программе, если количество повторений цикла велико, рекомендуется выполнять 2-3 повторения, включая последний оборот цикла, так как основная цель ручного исполнения – понять логику исполнения программы. Для удобного исполнения программы страницу тетради можно перевернуть, можно изготовить специальные дидактические пособия.

При использовании технологии имитационного моделирования исполнения программ компьютером происходит умственное развитие учащихся, формирование приёмов умственной деятельности, в частности, приёмов воображения. Учащиеся активизируют знания о функциональном назначении компонентов, из которых состоит компьютер, под руководством учителя овладевают способом имитации исполнения программы ПК – моделированием памяти ПК, далее исполняют программы с использованием наглядных протоколов, представляя мысленно схему прохождения информации в ПК при исполнении команд программы. Сначала учащиеся овладевают наглядным приёмом исполнения программы, затем они постепенно обучаются переносу наглядных приёмов в мысленную сферу. Можно предложить учащимся, при изучении учебного материала предмета «Информатика и ИКТ», выполнять проекты на тему «Имитационное моделирование», используя для имитации исполнения программы ПК про-

граммные средства PowerPoint, QBasic, Turbo Pascal, Visual Basic.Net, Turbo Delphi и др., смотрите приложения 1, 2.

Примеры ручного исполнения программ с использованием метода наглядных протоколов

Пример 6. Напишите программу решения квадратного уравнения $ax^2 + bx + c = 0$, ($a \neq 0$). Исполните составленную программу, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов, для $a = 1$, $b = -5$, $c = 6$ и $a = 1$, $b = -4$, $c = 4$. Пусть программа имеет имя – «*квур*».

Решение

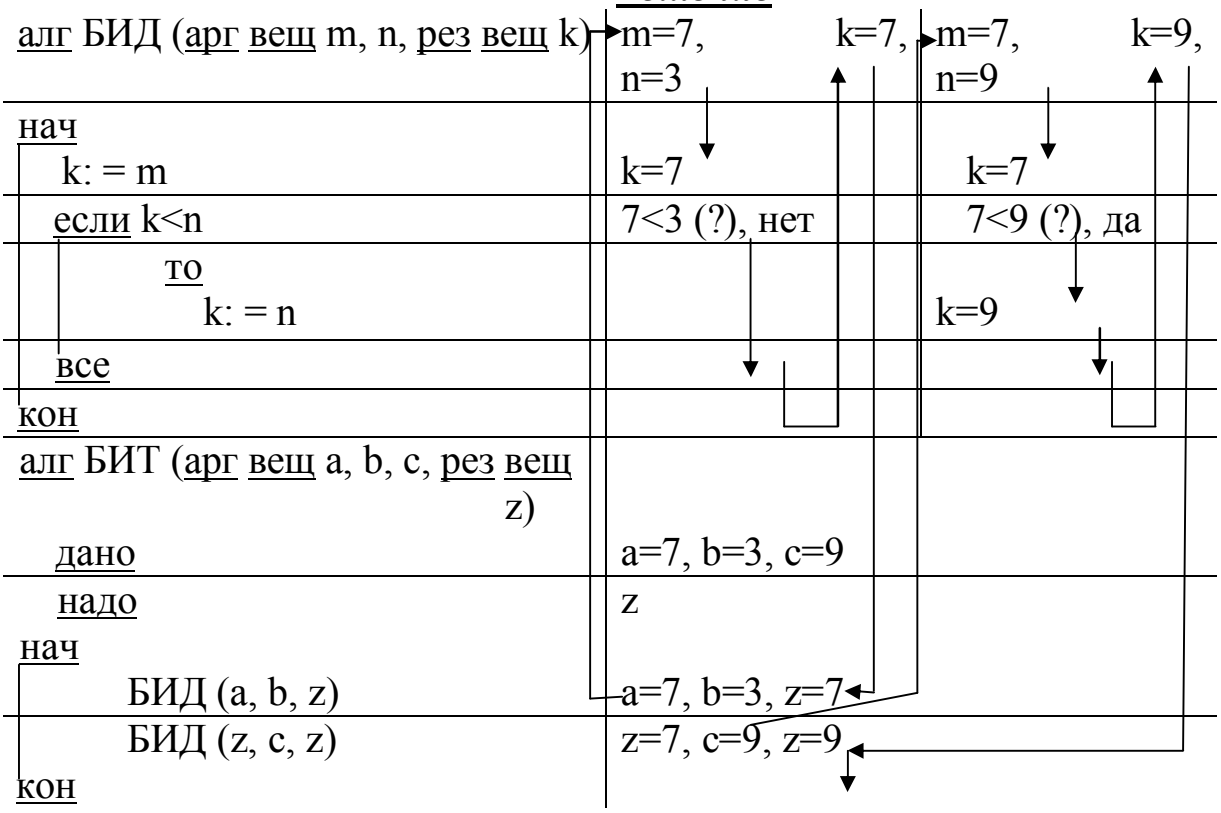
алг *квур* (арг вещ a, b, c, рез вещ x1, x2, рез лит y) дано	**a=1, b=-5, c=6**	**a =1, b=-4, c=4**
надо	x1, x2, y	x1, x2, y
нач вещ D		
D: = b*b – 4*a*c	**D=1**	**D= 0**
если D>0	**1>0 (?), да**	**0>0 (?), нет**
то		
y: = “2 разл. действит. корня”	y= «2 разл. дейст-вит. корня»	
x1: = (- b + **sqrt** (D))/(2*a)	**x1 = 3**	
x2: = (- b – **sqrt** (D))/(2*a)	**x2 = 2**	
иначе		
если D = 0		**0=0 (?), да**
то		
y: = “2 равных действ. корня”		**y=«2 равных действ.корня»**
x1: = - b/(2×a)		**x1 = 2**
x2: = x1		**x2 = 2**
иначе y: = “действ.корней нет”		
все		
все		
кон		

Пример 7. Напишите программу нахождения большего из трёх вещественных чисел, при составлении программы воспользуйтесь

вспомогательным алгоритмом вычисления большего из двух чисел. Исполните составленную программу, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов, для $a = 7$, $b = 3$, $c = 9$.

Пусть программа нахождения большего из двух вещественных чисел имеет имя «*БИД*», а большего из трёх – «*БИТ*».

Решение



Пример 8. Напишите программу для нахождения суммы элементов таблицы вещественных чисел a(1:*n*). Исполните составленную программу, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов, для $n = 3$ и a[1] = 4.2, a[2] = -3, a[3] = 5.

Решение

алг сумма (арг цел n, арг вещ таб a[1:n], рез вещ S)	
дано	n = 3, a[1] = 4.2, a[2] = -3, a[3] = 5

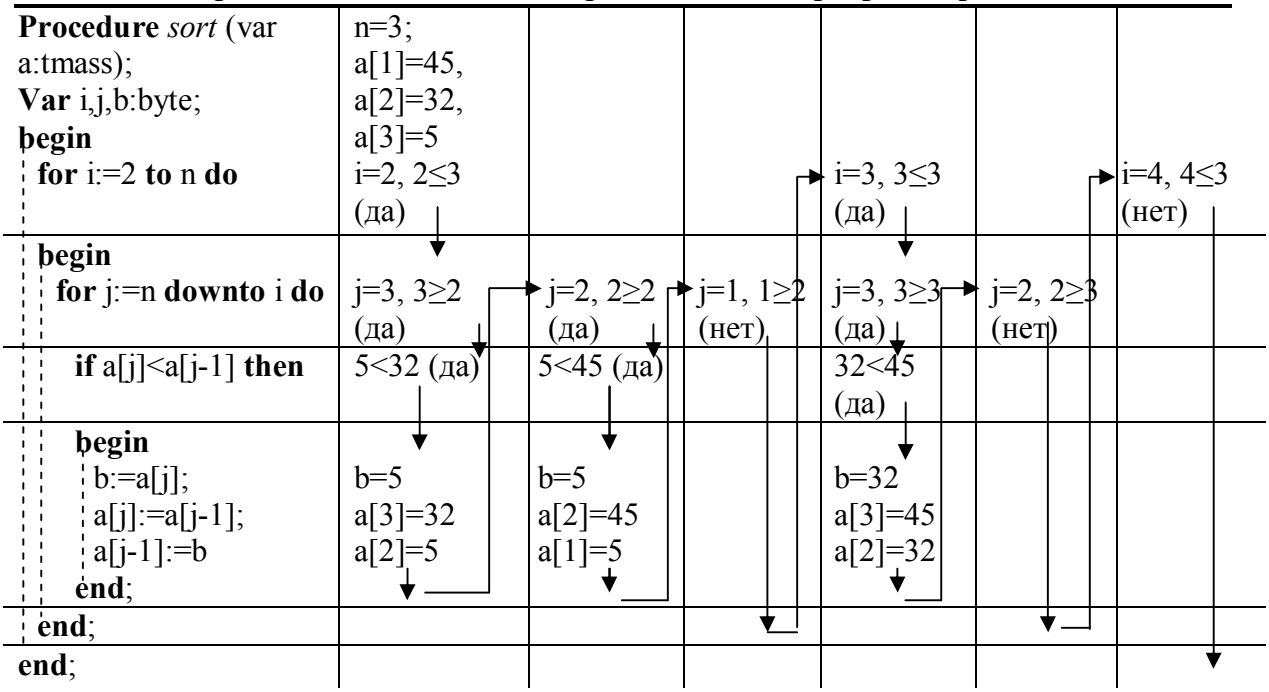
надо	S			
нач цел i				
S: = 0	S=0			
i: = 1	i=1			
нц пока i<=3	1<=3(?), да	2<=3?, да	3<=3?, да	4<=3?, нет
S: = S + a[i]	S=4.2	S=1.2	S=6.2	
i: = i + 1	i=2	i=3	i=4	
кц				
кон				

Пример 9. Напишите программу сортировки элементов массива целых чисел методом простого обмена. Исполните процедуру (*sort*) сортировки элементов массива простым обменом, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов, для n = 3 и a[1] = 45, a[2] = 32, a[3] = 5. Сущность метода сортировки элементов массива простым обменом изложена ниже, стр.180.

```
Program BubbleSort;
Uses crt;
Const n=3;
Type tmass=array[1..n] of byte;
Var a:tmass;
Procedure print (a: tmass);
{Процедура печати элементов масси-
ва}
Var  i:byte;
begin
   for i:=1 to n do write (a[i]:4);
end;
Procedure input (var a:tmass);
{Процедура ввода элементов массива}
Var  i:byte;
begin
   for i:=1 to n do
      begin
         write('a[',i, ']= '); readln(a[i]);
      end;
end;
```

```
Procedure sort (var a:tmass);
{Процедура сортировки обменом}
Var i,j,b:byte;
begin
   for i:=2 to n do
     begin
        for j:=n downto i do
          if a[j]<a[j-1] then
            begin
              b:=a[j];
              a[j]:=a[j-1];
              a[j-1]:=b
            end;
        print(a);
     end;
end;
Begin
   input (a);   sort (a);   print (a);
end.
```

Программа сортировки элементов массива простым обменом



Пример 10. Напишите программу замены в данном тексте *слова1* на *слово2*, той же длины. Исполните составленную программу, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов.

Решение. (Задача на вырезку и частичную замену). Пусть заданы строковой операнд *strText* (текст*)* и два слова: заменяемое слово *Word1* (слово1) и заменяющее слово *Word2* (слово2). Просматривая текст *strText*, начиная с первого символа (*i* = 1), *вырезаем* из текста *strText* последовательность из *d* символов, где *d* – длина *Word1* (*Word2*), сравниваем вырезку со словом *Word1*. Если они равны, то осуществляем *частичную замену*, а именно, последовательность из *d* символов в тексте *strText* заменяем на слово *Word2*. Увеличиваем *i* на *d* и, начиная с полученного *i*, вырезаем из текста *strText* следующие *d* символов. Если вырезка из *d* символов текста *strText* не совпадает со словом *Word1*, то *i* увеличиваем на 1 и вырезаем из текста *strText* следующие *d* символов. Вырезку, сравнение и, если выполнено условие, замену проводим пока *i* не станет больше длин(*текст*)-*d*+1.

Программа в системе Кумир, где определена операция частичной замены в тексте

```
алг замена (арг рез лит текст, арг лит слово1, слово2)
 дано |литерный операнд текст
```

```
 надо |в литерном операнде текст заменить слово1 на слово2
нач цел i,d
 | d:=длин(слово1)                  |находим длину слова1
 | i:=1          |организуем просмотр текста с первого символа
 | нц пока i<=длин(текст)-d+1
 | | если текст[i:i+d-1]=слово1  | текст[i:i+d-1]  – вырезка из текста
 | | |последовательности символов длины d и ее сравнение со словом1
 | | то текст[i:i+d-1]:=слово2  |частичная замена в тексте
 | | |  |последовательности из d символов на слово2, если вырезанный
 | | |   i:=i+d-1   | текст равен слову1, i увеличивается на d-1
 | | все
 | | i:=i+1  |номер следующего символа  в тексте
 | кц
кон
```

Программа в системе Кумир, где не определена операция частичной замены в тексте

```
алг замена1 (арг лит текст, арг лит слово1, слово2, рез лит текст1)
    дано |литерный операнд текст
    надо |в литерном операнде текст заменить слово1 на слово2
нач цел i,d
 | d:=длин(слово1) |находим длину слова1
 | i:=1; текст1:="" |результат формируем в текст1,
 | | сначало текст1 присвоим значение""
 | нц пока i<=длин(текст)-d+1
 | | если текст[i:i+d-1]=слово1
 | | | то текст1 := текст1 + слово2   |если условие верно,
 | | |    | то текст1 и слово1 склеиваем, и увеличиваем i на d-1
 | | |    i:=i+d-1
 | | | иначе
 | | |    текст1:=текст1 + текст[i:i]  |иначе текст1 склеиваем
 | | |    | с вырезкой из текст
 | | все
 | | i:=i+1 |номер следующего символа в тексте
 | кц
кон
```

Исполнение составленной программы «замена» методом моделирования в виде наглядных протоколов дано в таблице 4.

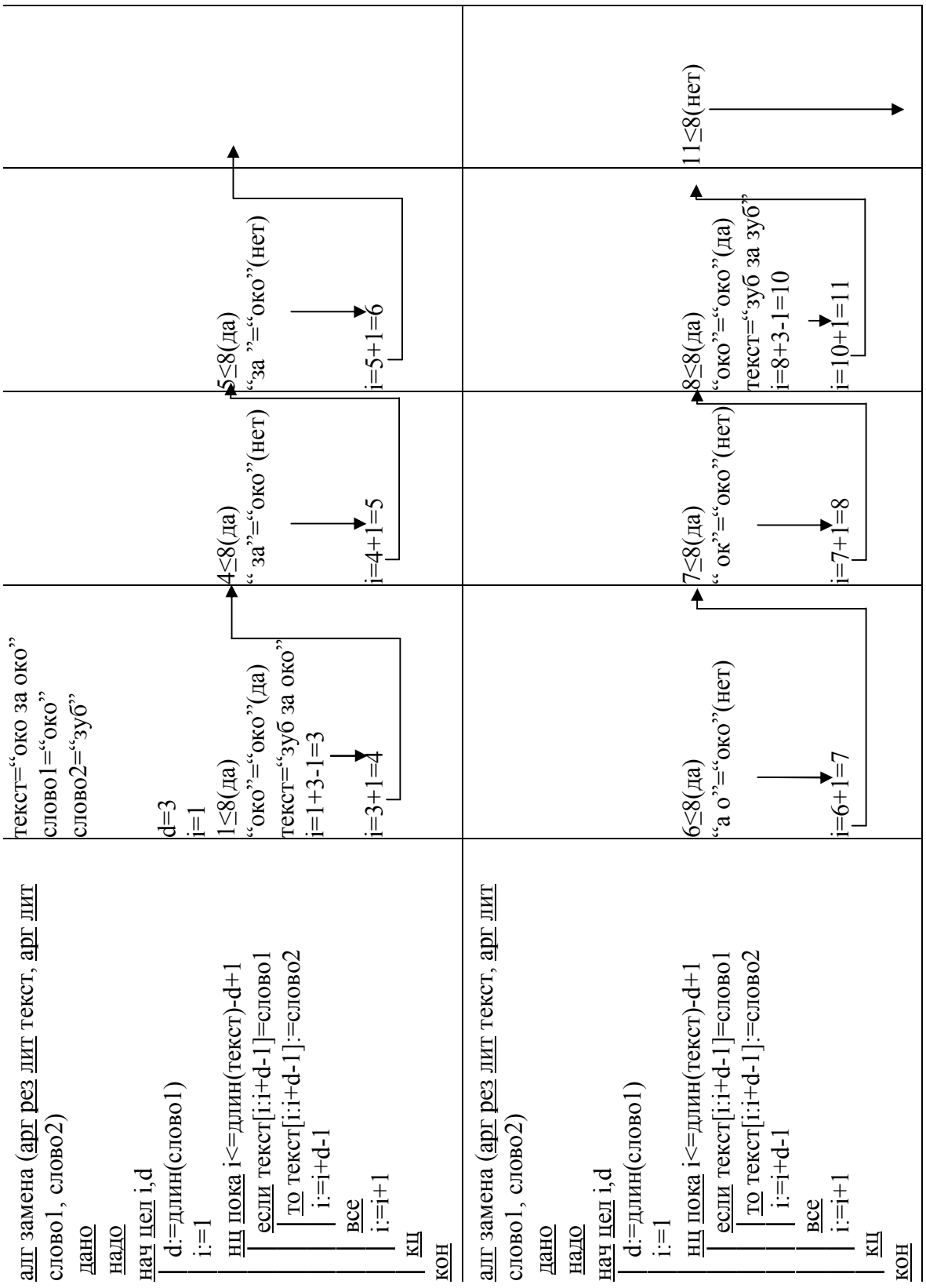

алг замена (арг рез лит текст, арг лит слово1, слово2)
дано
надо
нач цел i,d
d:=длин(слово1)
i:=1
нц пока i<=длин(текст)-d+1
если текст[i:i+d-1]=слово1
то текст[i:i+d-1]:=слово2
i:=i+d-1
все
i:=i+1
кц
кон
текст="око за око"
слово1="око"
слово2="зуб"
d=3
i=1
1≤8(да)
"око"="око"(да)
текст="зуб за око"
i=1+3-1=3
i=3+1=4
4≤8(да)
" за"="око"(нет)
i=4+1=5
5≤8(да)
"за "="око"(нет)
i=5+1=6
6≤8(да)
"а о"="око"(нет)
i=6+1=7
7≤8(да)
" ок"="око"(нет)
i=7+1=8
8≤8(да)
"око"="око"(да)
текст="зуб за зуб"
i=8+3-1=10
i=10+1=11
11≤8(нет)

Таблица 4

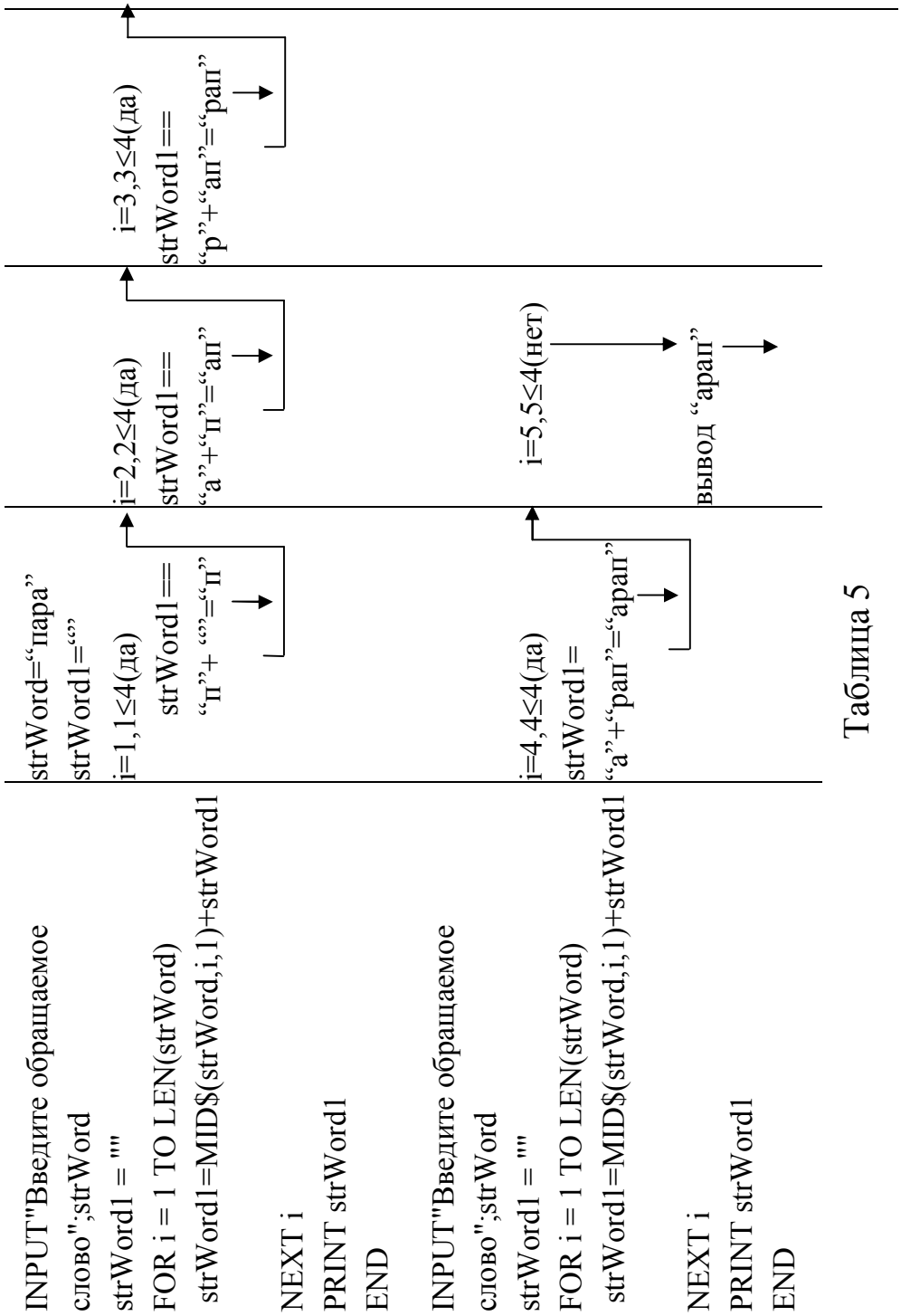

INPUT"Введите обращаемое слово";strWord
strWord1 = ""
FOR i = 1 TO LEN(strWord)
strWord1=MID$(strWord,i,1)+strWord1
NEXT i
PRINT strWord1
END
strWord="пара"
strWord1=""
i=1,1≤4(да)
strWord1== "п"+ ""="п"
i=2,2≤4(да)
strWord1== "а"+"п"="ап"
i=3,3≤4(да)
strWord1== "р"+"ап"="рап"
i=4,4≤4(да)
strWord1= "а"+"рап"="арап"
i=5,5≤4(нет)
вывод "арап"

Таблица 5

Пример 11. Напишите программу обращения слова. В результате исполнения алгоритма из слова «нас» должно получиться слово «сан», из слова «довереп» – слово «перевод». Исполните составленную программу, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов для слова «пара».

Решение. Пусть *strWord* – данное слово, а *strWord1* – получаемое слово, первоначально *strWord1* – пустая строка. Выделяем из строки *strWord* последовательно, по одному символу (с первого до последнего). Склеиваем вырезанный символ с получаемой строкой *strWord1* (*strWord1*:=”выделенный символ”+*strWord1*). Тогда первый выделенный символ из слова *strWord* будет последним символом в получаемом слове *strWord1*, второй выделенный символ из слова *strWord* – предпоследним символом в слове *strWord1* и т. д.

QBasic

```
REM обращение слова
DIM strWord AS STRING, strWord1 AS STRING, intI AS INTEGER
′описание операндов
INPUT "Введите обращаемое слово"; strWord       ′ввод strWord
strWord1 = ""              ′обращенное слово сначала равно пустой строке
FOR intI = 1 TO LEN(strWord) ′организуем вырезку последовательно
                             ′ всех символов строки strWord
  strWord1 = MID$(strWord, intI, 1) + strWord1       ′склеиваем строку
′strWord1 с вырезанным символом строки strWord
NEXT intI
PRINT strWord1             ′вывод обращенной строки strWord1 на экран
END
```

Ручное исполнение программы, написанной на языке QBasic, приведено в таблице 5, с. 168.

Пример 12. Напишите программы ввода и вывода элементов множества *SetA*. Исполните составленные программы, используя метод моделирования исполнения программы компьютером в виде наглядных протоколов для множества [8, 6, 9].

Решение

Программы *ввода и вывода* элементов *множества:*

```
Program TaskSet;
   Uses Crt;
   Type
      TSet=set of 5..9;
   Var
```

```
Program TaskSet1;
   Uses Crt;
   Type
      TSet=set of 5..9;
   Var
```

```
        i, n: byte;
        SetA: TSet;
        next: 5..9;
Begin
    ClrScr;
    {ввод элементов множества SetA}
    SetA:=[];
    Write('Задайте количество элементов множества');
    Readln(n);
    for i:=1 to n do
        begin
           Write(i,' – элемент?');
            Readln(next);
            SetA:=SetA+[next];
        end;
                    {вывод элементов множества}
                        for i:=5 to 9 do
                      if i in SetA then Write(i:4);
                End.
```

```
        i, n: byte;
        SetA: TSet;
        next: 5..9;  sim:char;
Begin
    ClrScr;
    {ввод элементов множества SetA}
    SetA:=[]; i:=1;
    Repeat
        Write(i,' – элемент?');
        Readln(next); i:=i+1;
        SetA:=SetA+[next];
        Write('Вводить элемент множества д/н?');
        Readln(sim)
    Until sim='н';
```

Ручное исполнение программ TaskSet и TaskSet1 дано в таблицах 6-8, с. 171.

Задания к данному параграфу

Выберите по две задачи, отмеченных (*), каждой главы работы [18]. Решите задачи. Исполните составленные программы, используя метод ручного моделирования исполнения программы компьютером в виде наглядных протоколов, для составленных вами тестов.

…
SetA:=[];
Write('Задайте кол-во элементов множества');
Readln(n);
for i:=1 to n do
begin
Write(i,' – элемент?');
Readln(next);
SetA:=SetA+[next];
end;
	SetA=[];
	На экране: Задайте количество элементов множества
	n=3;
	i=1, 1≤3(да)
	На экране: 1 – элемент?
	next=8
	SetA=[]+[8]=[8]
		i=2, 2≤3(да)
		На экране: 2 – элемент?
		next=6
		SetA=[8]+[6]=[8,6]
			i=3, 3≤3(да)
			На экране: 3 – элемент?
			next=9
			SetA=[8,6]+[9]=[8,6,9]
				i=4, 4≤3(нет)

Таблица 6. Ввод элементов множества, программа TaskSet

… SetA:=[]; i:=1; Repeat Write(i,' – элемент?'); Readln(next); i:=i+1; SetA:=SetA+[next]; Write('Вводить эл-т множества д/н?'); Readln(sim) Until sim='н';	SetA:=[]; i:=1 На экране:1 – элемент? next=8; i=1+1=2 SetA=[]+[8]=[8] На экране: Вводить элемент множества д/н? sim='д' 'д'='н' (нет)	На экране:2 – элемент? next=6; i=2+1=3 SetA=[8]+[6]=[8,6] На экране: Вводить элемент множества д/н? sim='д' 'д'='н' (нет)	На экране:3 – элемент? next=9; i=3+1=4 SetA=[8,6]+[9]=[8,6,9] На экране: Вводить элемент множества д/н? sim='н' 'н'='н' (да)

Таблица 7. Ввод элементов множества, программа TaskSet1

for i:=5 to 9 do
if i in SetA then
Write(i:4);
	i=5, 5≤9(да)
	5∈[8,6,9](нет)
		i=6, 6≤9(да)
		6∈[8,6,9](да)
		На экране:
		6
			i=7, 7≤9(да)
			7∈[8,6,9](нет)
				i=8, 8≤9(да)
				8∈[8,6,9](да)
				На экране:
				6 8
					i=9, 9≤9(да)
					9∈[8,6,9](да)
					На экране:
					6 8 9
						i=10,10≤9(нет)

Таблица 8. Вывод элементов множества

1.3. Имитационное моделирование при нахождении алгоритма поиска минимального элемента в массиве чисел

Основная цель обучения – развитие учащегося. Возможности развития учащихся таятся в соответствующем образом отображенном содержании учебного материала и составленных на этой основе познавательных заданий. Однако эти средства являются лишь предпосылкой развития. Для того, чтобы обучение имело развивающий эффект, нужно соблюдать следующее условие – *развиваемый субъект должен быть включен в активную деятельность и общение*. Содержательной стороной активизации учащихся являются задания на активизацию памяти, процесса логического мышления, на творческую деятельность, на поиск новых знаний. Тема «Поиск в линейной таблице элемента, обладающего заданным свойством» посвящена изучению процесса нахождения данных с определенной идентификацией. Например, в вычислительной задаче нам может понадобиться найти f(x), имея x и таблицу значений f; в лингвистической – может интересовать английский перевод данного русского слова. Существует много различных видов поиска: линейный поиск, бинарный поиск, метод золотого сечения и др. В школьном курсе информатики изучают два метода поиска: линейный и бинарный. Рассмотрим некоторые *методические аспекты* активации логического мышления, творческой деятельности, «*открытия новых знаний»* при изучении этой темы в школьном предмете «Информатика и ИКТ».

Задача поиска минимального элемента ставится следующим образом. Имеется линейный массив (таблица) вещественных чисел A[t:n], требуется найти номер k элемента A[k], который имеет в массиве минимальное значение.

A[1]	A[2]	A[3]	A[4]
1	2	3	4
10	8	14	6

Для разъяснения идеи построения алгоритма полезно обратиться к конкретному примеру. Для *осознанного* получения новых знаний мы предлагаем при составлении алгоритма использовать моделирование памяти компьютера. Пусть имеется линейный массив, состоящий

из четырех элементов. Запишем имена элементов массива, их индексы и значения в клетки таблицы. В первой строке таблицы запишем имена элементов массива, во второй строке – индексы элементов массива, в третьей строке – значения соответствующих элементов массива. Выберем две вспомогательные переменные: *k* – для запоминания номера минимального элемента, *min*1 – для запоминания значения этого элемента. Имена и значения переменных *k* и *min*1 для наглядности также будем записывать в особые клетки.

1 шаг. Предположим, что минимальным элементом массива является его первый элемент. Поместим в клетку *k* номер первого элемента массива (1), в клетку *min*1 – значение первого элемента массива (10).

k	***min*1**		A[1]	A[2]	A[3]	A[4]
			1	2	3	4
1	10		10	8	14	6

Далее будем сравнивать все значения элементов массива, начиная со второго, со значением переменной *min*1, т.е. со значением, записанным в клетке *min*1. Если при этом окажется, что сравниваемый с *min*1, элемент A[*i*], где *i* изменяется от 2 до 4, меньше, чем значение *min*1, то переменной *min*1 присвоим значение A[*i*], а переменной *k* – номер элемента A[*i*] (*i*). Если же элемент A[*i*] окажется больше *min*1, то значения переменных *min*1 и *k* будем оставлять без изменения.

2 шаг. *i* = 2, сравниваем A[2] с *min*1. A[2] = 8, *min*1 = 10. A[2] меньше *min*1? Да, так как 8 < 10. Следовательно, *min*1 нужно присвоить значение A[2], а переменной *k* – значение 2, *min*1:= A[2], *k*:=2. В клеточку *min*1 записываем значение 8, а в клеточку *k* значение 2.

i=2
A[2]<*min*1 (да)
*min*1 := A[2]
k:=2

k	***min*1**		A[1]	A[2]	A[3]	A[4]
			1	2	3	4
2	8		10	8	14	6

3 шаг. *i* = 3, сравниваем A[3] с *min*1. A[3] = 14, *min*1 =8. A[3] меньше *min*1? Нет, так как 14 > 8, значения *min*1 и *k* оставляем без изменения.

i=3
A[*i*]<*min*1 (нет)
(ничего не делаем)

k	***min*1**		A[1]	A[2]	A[3]	A[4]
			1	2	3	4
2	8		10	8	14	6

4 шаг. *i* = 4, сравниваем A[4] с *min*1. A[4] = 6, *min*1 = 8. A[4] меньше *min*1? Да, так как 6 < 8. Следовательно, *min*1 нужно присвоить значение A[4], а переменной *k* – значение 4, *min*1:=A[2], k:=4. В клеточку *min*1 записываем значение 6, а в клеточку *k* значение 4.

i=4
A[*i*]<*min*1 (да)
*min*1 := A[*i*]
k:=*i*

k	***min*1**		A[1]	A[2]	A[3]	A[4]
			1	2	3	4
4	6		10	8	14	6

Итак, минимальный элемент массива A[1:4] – это четвёртый элемент, его значение 6.

Пример программы поиска минимального элемента в последовательности целых чисел – элементов массива, начиная с номера t до номера r.

```
алг поиск минимального (арг цел n,t,r)
нач цел таб a[1:n],цел i, цел k,цел min1
| input(n,a); print(n,a)
|поиск k-индекса минимального
|элемента и min1- значения
|минимального элемента
|в массиве, начиная
|с номера t до номера r
| min1:=a[t]; k:=t
| нц для i от t+1 до r
| | если min1>a[i]
| | | то
| | |   min1:=a[i]; k :=i
| | все
| кц
| |вывод k и min1
| вывод нс, "k= ",k," ","min1= ", min1
Кон
```

```
алг print (арг цел n,арг цел таб a[1:n])
нач цел i
| |вывод элементов массива
| вывод нс
| нц для i от 1 до n
| | вывод a[i]," "
| кц
кон
алг input (арг цел n, рез цел таб a[1:n])
нач цел i
| |ввод элементов массива
| нц для i от 1 до n
| | вывод "a[",i,"]= "
| | ввод a[i]
| кц
кон
```

Программа поиска минимального элемента массива на языке Ершол, система Кумир

Задания к данному параграфу

Задание 1. Задайте массив вещественных чисел a[1:*n*]. Смоделируйте последовательность преобразований содержимого памяти при

реализации метода поиска элемента массива, удовлетворяющего заданному требованию:

- найдите минимальный, максимальный, равный заданному значению, элемент массива и его индекс;
- найдите номер первого элемента массива, равного заданному с клавиатуры числу *b*, т. е. наименьшее *i*, при котором a[*i*]=*b*;
- найдите наибольшее *i*, при котором a[*i*] >0;

Задание 2. Составьте блок-схемы алгоритмов решения задач *задания 1*, напишите программы на выбранных языках программирования.

1.4. Имитационное моделирование при нахождении алгоритма сортировки элементов массива методом выбора

Сортировка выбором

Рассмотрим *сортировку простым выбором.* Задача сортировки элементов массива достаточно сложная для учащихся задача алгоритмизации. Метод упорядочения элементов линейного массива методом выбора опирается на задачу поиска минимального (максимального) элемента массива. Задачу поиска минимального (максимального) элемента массива учащиеся должны рассмотреть при изучении темы «Поиск заданного элемента в массиве».

Прежде чем переходить к формулировке алгоритмов в терминах алгоритмического языки, нужно рассмотреть основную идею, на которой основывается алгоритм упорядочения. Суть используемого приема состоит в последовательном рассмотрении ряда массивов A[1:*n*], A[2:*n*], A[3:*n*], A[4:*n*], …, A[*n*-1:*n*], составленных из элементов исходного массива, каждый из которых содержит на один элемент меньше, чем предыдущий. В каждом из этих массивов отыскивается наименьший элемент, который тут же переставляется местами с первым элементом рассматриваемого массива (первый раз – с A[1], второй раз – с A[2], третий – с A[3], … , последний раз – с A[*n*-1]). Процесс заканчивается, когда остаётся один элемент исходного массива. Понятно, что совокупность, не рассматриваемых на каждом этапе элементов массива, образует в результате массив, упорядоченный по возрастанию. Используемый прием лучше всего пояснить на конкрет-

ном примере, *моделируя память компьютера*. Пусть имеется линейный массив A[1:5], состоящий из пяти элементов:

A[1]	A[2]	A[3]	A[4]	A[5]
1	2	3	4	5
24	19	15	18	22

В первой строке таблицы записаны имена элементов массива. Во второй строке – индексы элементов массива. В третьей строке – значения соответствующих элементов массива. Выберем в таблице минимальный элемент, это элемент A[3] =15, и переставим его с первым элементом, A[1] = 24, A[1] ↔ A[3]. Получим таблицу:

A[1]	A[2]	A[3]	A[4]	A[5]
1	2	3	4	5
15	19	24	18	22

Отставим теперь в сторону первый элемент массива, он стоит на своём месте, и будем рассматривать новый массив A[2:5], образованный из прежнего, отбрасыванием первого элемента:

A[1]
1
15

A[2]	A[3]	A[4]	A[5]
2	3	4	5
19	24	18	22

Найдем теперь минимальный элемент в новом массиве A[2:5], это элемент A[4] = 18, переставим его со вторым элементом, A[2] = 19, A[2] ↔ A[4]. Получим таблицу:

A[1]
1
15

A[2]	A[3]	A[4]	A[5]
2	3	4	5
18	24	19	22

Элемент A[2] массива стоит на своём месте в получаемом отсортированном массиве, отставим его в сторону:

A[1]	A[2]
1	2
15	18

A[3]	A[4]	A[5]
3	4	5
24	19	22

Найдем теперь минимальный элемент в новом массиве A[3:5], это элемент A[4] = 19, переставим его с третьим элементом массива, A[3] = 24, A[3] ↔ A[4]. Получим таблицу:

A[1]	A[2]
1	2
15	18

A[3]	A[4]	A[5]
3	4	5
19	24	22

Элемент A[3] массива стоит на своём месте в получаемом отсортированном массиве, отставим его в сторону.

A[1]	A[2]	A[3]
1	2	3
15	18	19

A[4]	A[5]
4	5
24	22

Найдем теперь минимальный элемент в массиве A[4:5], это элемент A[5] = 22, переставим его с четвёртым элементом массива, A[4] = 24, A[4] ↔ A[5]. Получим таблицу:

A[1]	A[2]	A[3]
1	2	3
15	18	19

A[4]	A[5]
4	5
22	24

Элемент A[4] массива стоит на своём месте в получаемом отсортированном массиве, отставим его в сторону.

A[1]	A[2]	A[3]	A[4]
1	2	3	4
15	18	19	22

A[5]
5
24

Последний элемент A[5] массива при данной сортировке всегда стоит *на своём месте* в получаемом отсортированном массиве, его тоже можно отставить влево.

Получим *окончательное решение* в виде такой таблицы:

A[1]	A[2]	A[3]	A[4]	A[5]
1	2	3	4	5
15	18	19	22	24

Третья строка таблицы – элементы отсортированного массива.

При рассмотрении примера, видно, что вполне самостоятельную часть алгоритма упорядочения элементов массива этим методом, составляет ***алгоритм нахождения минимального элемента*** в массиве A[*k*:*n*], где *k* изменяется от 1 до *n*-1. Можно сделать вывод, что алгоритм поиска минимального элемента в массиве A[*k*:*n*] можно рассматривать как вспомогательный по отношению к основному алгоритму упорядочения массива.

Пример программы сортировки элементов массива методом простого выбора

```
алг сортвыбор (арг цел n)
нач цел i,k,b,цел таб a[1:n]
| input(n,a)
| print(n,a)
| нц для i от 1 до n-1
| | индмин (i,n,a,k)
| | |обмен значениями a[i] и a[k]
| | b:=a[i]; a[i]:=a[k];a[k]:=b
| | print(n,a)
| кц
кон
алг индмин (арг цел i,n,арг цел таб
a[1:n],рез цел k)
нач цел min1,j
| |поиск k – индекса минимального
| |элемента в массиве, начиная
| |с номера i до номера n
| min1:=a[i]; k:=i
| нц для j от i+1 до n
| | если min1>a[j]
| | | то
| | |   min1:=a[j]
| | |   k:=j
| | все
| кц
кон
```

```
алг print (арг цел n,арг цел таб a[1:n])
нач цел i
| |вывод элементов массива
| вывод нс
| нц для i от 1 до n
| | вывод a[i]," "
| кц
кон
алг input (арг цел n,рез цел таб a[1:n])
нач цел i
| |ввод элементов массива
| нц для i от 1 до n
| | вывод "a[",i,"]= "
| | ввод a[i]
| кц
кон
```

Таблица 9. Программа сортировки элементов массива методом простого выбора на языке Ершол

При составлении программы сортировки элементов массива методом простого выбора применим технологию проектирования алгоритма «*снизу вверх*»**.** При рассмотрении примера было выяснено, что для получения результата необходимо n-1 раз последовательно находить индекс k минимального элемента массивов A[1:n],A[2:n], A[3:n], A[4:n], … , A[n-1:n], составленных из элементов исходного массива, каждый из которых содержит на один элемент меньше, чем предыдущий. Далее элемент A[k] тут же необходимо переставить местами с первым элементом рассматриваемого массива (первый раз с – A[1], второй раз – с A[2], третий – с A[3], … , последний раз – с A[n-1]).

Процесс заканчивается, когда остаётся один элемент исходного массива. Алгоритм сортировки можно записать так:

алг *сортвыбор*
нач
1шаг. Введи элементы массива в память компьютера.
2 шаг. Для *i,* изменяющегося от 1 до *n*-1, выполни следующие шаги.
2.1 шаг. Найди номер *k* минимального элемента массива *a*[*i*:*n*].
2.2 шаг. Поменяй местами a[*i*] и a[*k*].
2.3 шаг. Организуй вывод элементов массива, полученных при прохождении текущего цикла.
кон

Для составления программы, реализующей шаги 1 и 2.3 алгоритма, воспользуемся подпрограммами, составленными на странице 174.

Замечание: Процедура на Turbo Delphi, имитирующая механизм исполнения сортировки методом простого выбора, представлена в приложении 2.

Задания к данному параграфу

Задание 1. Задайте массив целых чисел a[1:*n*]. Смоделируйте последовательность преобразований содержимого памяти при реализации *сортировки* элементов массива *методами:*

- *методом простого выбора;*
- *методом простых включений,* стр. 147, 149,151;
- *методом простого обмена, суть метода изложена ниже;*
- *улучшенным методом сортировки обменом,* когда необходимо запомнить, производился ли на данном проходе массива обмен элементов, если нет, то сортировку массива закончить*;*
- *улучшенным методом сортировки обменом,* когда необходимо запомнить не только, производился ли на данном проходе массива обмен элементов, но и место последнего обмена, следующие проходы можно закончить на этом индексе, вместо того, чтобы сравнивать элементы до установленной границы *i*.

Задание 2. Составьте блок-схемы алгоритмов решения задач задания 1, напишите программы на языках программирования, отличных от Ершола.

Пояснение к выполнению задания

Идея метода сортировки массивов простым обменом

Идея сортировки простым обменом элементов массива a[1:*n*] заключается в повторении *n*-1 раз попарных сравнений рядом стоящих элементов массивов A[1:*n*], A[2:*n*], A[3:*n*], A[4:*n*], … , A[*n*-1:*n*]. Каждый из этих массивов содержит на один элемент меньше, чем предыдущий. При сравнении элементы массива переставляются в заданном порядке, поэтому при каждом проходе элемент a[*i*], *i* изменяется от 1 до *n*-1, занимает своё место в отсортированной части массива. Последний элемент массива после попарных сравнений *n*-1 раз и обменов, если необходимо, стоит тоже на своём месте в отсортированном массиве. Элементы массивов просматриваются либо *от конца к началу,* либо *от начала к концу*. При сортировке будем просматривать элементы массивов A[1:*n*], A[2:*n*], A[3:*n*], A[4:*n*], … , A[*n*-1:*n*] *от конца к началу,* и массив отсортируем по *возрастанию* значений элементов массива. Алгоритм сортировки элементов массива методом простого обмена и программа на языке Ершол даны в таблице 10.

Программа сортировки массивов простым обменом и сущность этого метода

алг *сортобмен* нач 1шаг. Введи элементы массива в память компьютера. 2 шаг. Для *i* изменяющегося от 2 до *n* выполни следующие шаги. 2.1 шаг. Для *j,* изменяющегося от *n* до *i* с шагом -1, выполни следующие шаги. 2.1.1 шаг. Сравни элементы a[*j*] и a[*j*-1]. Если a[*j*] < a[*j*-1], то поменяй их местами. 2.1.2 шаг. Организуй вывод элементов массива, полученного при прохождении текущего цикла. кон	алг сортобмен (арг цел n) нач цел i,j,b,цел таб a[1:n] input(n,a) print(n,a) нц для i от 2 до n нц для j от n до i шаг -1 если a[j] < a[j-1] то |обмен значениями a[j] и a[j-1] b:=a[j]; a[j]:=a[j-1];a[j-1]:=b все кц print(n,a) кц кон Вспомогательные алгоритмы *input* и *print* смотри в таблице 9.

Таблица 10. Алгоритм и программа на языке Ершол сортировки элементов массива методом простого обмена

1.5. Имитационное моделирование при изучении механизма пирамидальной сортировки элементов массива

Задача 1. Разработайте алгоритм пирамидальной сортировки линейного массива и напишите программу на языке Turbo Pascal.

Решение

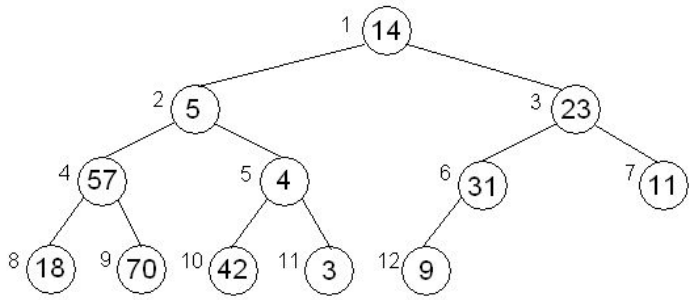
Алгоритм пирамидальной сортировки разработан Дж. Уильямсом и Р.У. Флойдом. Это *улучшенный метод* сортировки, основанный *на принципе выбора*. Методы сортировок используют многократные просмотры массивов и их частей и выполнение определенных операций над их элементами. Метод сортировки *простым выбором* базируется на *повторном выборе* наибольшего (наименьшего) элемента из n элементов, затем выбора наибольшего (наименьшего) элемента из n-1 элемента и т.д. Для улучшения метода сортировки простым выбором необходимо получать от каждого прохода больше информации, чем указание на один наибольший (наименьший) элемент. Алгоритм пирамидальной сортировки использует при выборе наибольшего (наименьшего) элемента представление сортируемого массива в виде нелинейной структуры – специального двоичного (бинарного) дерева (пирамиды).

Некоторые определения

Графом G называют *пару множеств* (W,E), где W – это непустое конечное множество элементов, которое мы назовем вершинами графа, а E – это конечное множество неупорядоченных или упорядоченных пар элементов множества W. Элементы множества E – ребра или дуги. Неупорядоченную пару вершин $\{V, U\}$ будем называть ребром, а упорядоченную пару назовем дугой $<V, U>$ (V – начало дуги, U – конец дуги). Две вершины U и V графа G называются *смежными,* если существует соединяющее их ребро $\{U, V\}$. При этом вершины U и V называются *инцидентными* этому ребру, а ребро *инцидентным* этим вершинам. Два ребра называются *смежными,* если они имеют, по крайней мере, одну общую вершину. *Цепью* в графе называется чередующаяся *последовательность вершин и ребер*: V_0, l_1, V_1, l_2 ,V_2 , … , l_k, V_k, в которой любые два соседних элемента *инцидентны* и все ребра различны. Если $V_0=V_k$, то цепь называется *циклом*. Если все вершины в цепи различны, то цепь называется *простой*. Если в графе любые две вершины соединены цепью, то граф называется *связным*. *Деревом* называется *связный граф без циклов.* При изображении графов на рисунках рёбра и дуги изображаются линиями, у дуг на

концах изображаются стрелки, указывающие направление связи вершин, вершины – точками, окружностями или квадратами. Такое представление графа называется диаграммой. *Диаграмма* графа, это *информационная модель* объекта, это информация о составе и структуре системы, представленная в графической форме. Отношения между вершинами дерева определяются в терминах взятых из генеалогической терминологии. *Вершина A*, которая находится над другой вершиной *C* и непосредственно соединена с ней ребром, называется *предком* вершины *C*. Если вершина *A* является предком вершины *C*, то *вершина C* называется *потомком* вершины *A*. *Корень* дерева – это вершина, *не имеющая предков* (первоначально выбранная вершина). *Листья* дерева – это вершины, *не имеющие потомков*. Вершина со своими потомками называется *поддеревом основного дерева*. *Бинарным деревом* называется дерево, в котором каждая вершина имеет *не более двух потомков.* В случае, когда вершина имеет два потомка, принято различать *левого* и *правого потомка*.

Возьмем массив *Mass* = (14, 5, 23, 57, 4, 31, 11, 18, 70, 42, 3, 9) и поставим в соответствие этому массиву бинарное дерево *B*. Каждому элементу массива поставим в соответствие вершину бинарного дерева, а отношения между элементами массива установим ребрами.



Уровень	Узлы
0	1
1	2, 3
2	4, 5, 6, 7
3	8, 9, 10, 11, 12

Натуральными числами начиная с 1 и далее «сверху вниз» по уровням и «слева направо» на одном уровне пронумеруем все вершины бинарного дерева *B*, а следовательно и все соответствующие элементы массива. Эти номера (адреса) поставлены около вершин. Корень дерева имеет адрес 1, содержимое корня – 14. Рассмотрим вершину с адресом 2, содержимое этой вершины – 5. Она имеет предка – вершину с адресом 1 и двух потомков – вершины с адресами 4 и 5. Вершина с адресом 6 имеет предка – вершину с адресом 3 и одного потомка – вершину с адресом 12. Вершина с адресом 7 имеет предка с адресом 3 и не имеет потомков. Вершины с адресами 8, 9, 10,

11, 12, 7 – листья (они не имеют потомков). Адреса предков и потомков вершины с адресом k можно вычислить по формулам:

- для предков pred(k) = $k\backslash 2$, где $k = 1,2,\ldots,n$;
- для потомка слева $\text{left}(k) = \begin{cases} 2*k, & k = 1,2,\ldots,n\backslash 2, \\ 0, & k > n\backslash 2; \end{cases}$
- для потомка справа $\text{right}(k) = \begin{cases} 2*k+1, & k = 1,2,\ldots,(n-1)\backslash 2, \\ 0, & k > (n-1)\backslash 2. \end{cases}$.

Введем понятие пирамиды. Пусть дан массив *Mass*:

$$\text{Mass}(1), \text{Mass}(2), \ldots, \text{Mass}(n). \qquad (1)$$

Пирамидой называется непустая последовательность элементов массива (1):

$$\text{Mass}(p), \text{Mass}(p+1), \ldots, \text{Mass}(q), \quad \text{где } 1 \le p \le q \le n,$$

для которой выполнено одно из следующих условий:

1) $2p > q$;
2) $2p = q$ и $\text{Mass}(p) \ge \text{Mass}(q)$;
3) $2p \le q$ и $\text{Mass}(j) \ge \text{Mass}(2j)$, для $p \le j \le q/2$,
   $\text{Mass}(j) \ge \text{Mass}(2j+1)$, для $p \le j \le (q-1)/2$.

Следствия:

1) для любой последовательности (1) подпоследовательность

$$\text{Mass}(n\backslash 2+1), \text{Mass}(n\backslash 2+2), \ldots, \text{Mass}(n)$$

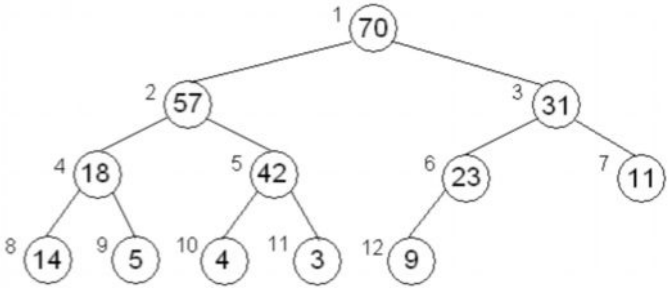
является пирамидой;

2) если последовательность (1) – пирамида, то

$$\text{Mass}(1) \ge \max_{1 \le j \le n} \text{Mass}(j);$$

3) если последовательность (1) – пирамида и представлена в виде бинарного дерева B, то значение любого узла в B будет не меньше значений его левого и правого потомков.

Пирамида, составленная из элементов массива *Mass*=(14, 5, 23, 57, 4, 31, 11, 18, 70, 42, 3, 9) – это последовательность (70, 57, 31, 18, 42, 23, 11, 14, 5, 4, 3, 9). Двоичное дерево, соответствующее пирамиде представлено на рисунке.



Алгоритм пирамидальной сортировки

алг пирамидальная_сортировка (арг цел n, арг рез вещ таб Mass[1:n])
дано || массив вещественных чисел Mass[1:*n*]
надо || отсортированный по возрастанию массив *Mass*
нач
| 1 шаг. Выбери наибольший элемент из *n* элементов массива *Mass* путем построения пирамиды из *n* элементов.
| 2 шаг. Поменяй местами первый и последний элементы массива *Mass*, поставив наибольший из *n* элементов на последнее место в массиве.
| 3 шаг. Выбери наибольший из первых *n*-1 элементов массива *Mass* путем построения из них пирамиды.
| 4 шаг. Поменяй местами первый и последний элементы просматриваемой части массива *Mass*, поставив наибольший элемент из первых *n*-1 элементов на *n*-1 место в массиве *Mass*.
| 5 шаг. Выбери наибольший из первых *n*-2 элементов массива *Mass* путем построения из них пирамиды.
| 6 шаг. Поменяй местами первый и последний элементы просматриваемой части массива *Mass*, поставив наибольший элемент из первых *n*-2 элементов на *n*-2 место в массиве *Mass*.
| и т.д.
кон

Итак, шаги 2 и 3 повторяются *n-1* раз для последовательно укорачивающегося справа на 1 элемент массива *Mass*. Реализация шага 1 отличается от реализации шагов 3, 5, 7 и т.д. На шаге 1 наибольший элемент выбирается путем построения пирамиды из всех элементов массива *Mass*: первоначально пирамида строится на основании следствия 1, далее последовательно к выбранной или построенной пирамиде добавляется слева элемент массива Mass(*k*), и при помощи просеивания добавленный элемент помещается на соответствующее место, чтобы снова получилась пирамида. Элементы добавляются и просеиваются, начиная с элемента Mass(*n*\2) до Mass(1).

Алгоритм пирамидальной сортировки можно переписать так:

```
алг пирамидальная_сортировка (арг цел n, арг рез вещ таб Mass[1:n])
 дано || массив вещественных чисел Mass[1:n]
 надо || отсортированный по возрастанию массив Mass
нач
│Построй пирамиду из n элементов массива Mass.
│нц  Повтори n-1 раз
│ │1. Поменяй местами первый и последний элементы построенной
│ │пирамиды из элементов массива Mass, и укороти для исследования
│ │просматриваемую часть массива справа на 1 элемент.
│ │2. Построй пирамиду из исследуемой части массива Mass путем
│ │просеивания только 1-го элемента.
│ кц
кон
```

Детализируем 1 шаг алгоритма – *построение пирамиды* из n элементов массива *Mass*. Выберем из заданного массива (1) последовательность, которая заведомо является пирамидой. По следствию 1 это последовательность:

$$Mass(n\backslash 2+1), Mass(n\backslash 2+2), \dots , Mass(n), \qquad (2)$$

т.к. выполнено условие $2*(n\backslash 2+1)\geq n$ или $2p>q$, где $p = n\backslash 2+1$ и $q = n$. Элементы последовательности (2) на бинарном дереве соответствуют листьям. Расширим пирамиду влево, добавляя к ней слева по одному оставшемуся элементу из массива (1) и, просеивая этот элемент по потомкам, поместим его на соответствующее место в наращенной пирамиде. Добавим к пирамиде (2) элемент $Mass(n\backslash 2)$ и обозначим $n\backslash 2$ через i. Будем иметь

$$Mass(i), Mass(i+1), \dots , Mass(n). \qquad (3)$$

Преобразуем (3) в пирамиду. Для этого сравним $Mass(i)$ с его потомками $Mass(2*i)$ и $Mass(2*i+1)$. Если $Mass(i)$ не меньше своих потомков, то просеивание элемента $Mass(i)$ заканчиваем, т.к. (3) – пирамида, в противном случае обмениваем значения $Mass(i)$ и $\max(Mass(2*i),Mass(2*i+1))$. Опустившийся элемент в последовательности (3) продолжаем просеивать тем же способом, пока последовательность (3) не станем пирамидой (смотри

иллюстрацию исполнения пирамидальной сортировки для конкретного массива). Продолжая наращивать пирамиду (3), преобразуем весь массив (1) в пирамиду. Назовем пирамиду текущей.

Детализируем шаг 2, который состоит из 2-х шагов: 2.1 и 2.2.

Шаг 2.1. В полученной пирамиде *Mass* первый элемент не меньше остальных, поставим его на последнее (*свое*) место, а последний элемент текущей пирамиды – на первое. Укоротим рассматриваемую часть массива на один элемент справа.

Шаг 2.2. Рассмотрим укороченную справа последовательность элементов, она уже может и не быть пирамидой. Для построения пирамиды необходимо просеять один первый элемент.

Далее шаги 2.1 и 2.2 повторить *n*-2 раза. В результате получим отсортированный по возрастанию массив.

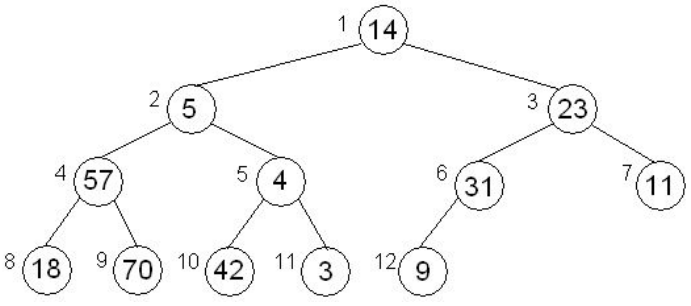
```
Program HeapSort;
   Uses Crt;
   Const byteN=12;
   Var  byteI,byteK:byte;  massA:real; Mass:array [1..byteN] of
real;
Procedure Sift (byteX,byteK:byte);
   Var byteY:byte;
Begin
   byteY:=2*byteX; {Нахождение индексов потомков элемента Mass[byteX]}
   While byteY<=byteK do {Организация просеивания элемента
Mass[byteX] по бинарному дереву для построения пирамиды}
      begin
         If byteY<byteK then
            {Выбор большего элемента последовательности из потомков
(Mass[byteY] и Mass[byteY]) элемента Mass[byteX]}
            If Mass[byteY]<Mass[byteY+1] then
               byteY:=byteY+1;
         {Выбор большего элемента из элемента Mass[byteX] и его потомков;
если элемент Mass[byteX] больше своих потомков, то дерево построено и
просеивание закончено, иначе переставляются элементы Mass[byteX] и
max(Mass[byteY],Mass[byteY+1]) и просеивание продолжается}
         If Mass[byteY]<=Mass[byteX] then break;
         massA:=Mass[byteX];  Mass[byteX]:=Mass[byteY];
         Mass[byteY]:=massA; byteX:=byteY;  byteY:=2*byteX
      end;
End;
BEGIN
   For byteI:=1 to byteN do {Ввод элементов сортируемого массива}
```

```
        begin
            Write('Mass[',byteI,']=');  ReadLn(Mass[byteI])
        end;
{Построение пирамиды из последовательности чисел}
    byteI:=(byteN div 2)+1; {Вычисление индекса первого элемента
                             последовательности для просеивания}
    While byteI>1 do
        begin
            byteI:=byteI-1;   Sift(byteI,byteN);
        end;
    byteK:=byteN;
    While byteK>1 do
        begin
            massA:=Mass[1]; {Перестановка первого и последнего элементов}
            Mass[1]:=Mass[byteK]; {исследуемой последовательности}
            mass[byteK]:=massA;
            byteK:=byteK-1; {Нахождение индекса последнего элемента
укороченной последовательности}
            Sift(1,byteK) {Просеивание первого элемента последовательности}
        end;
    For byteI:=1 to byteN do
        Write(Mass[byteI]:6:2)
END.
```

Иллюстрация исполнения пирамидальной сортировки для конкретного массива

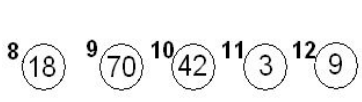
Дано:

Индексы	1	2	3	4	5	6	7	8	9	10	11	12
Значения	14	5	23	57	4	31	11	18	70	42	3	9
Таблица значений и индексов элементов массива *Mass*												



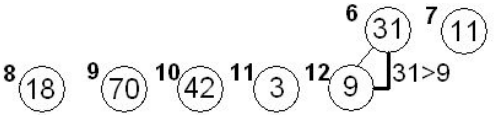
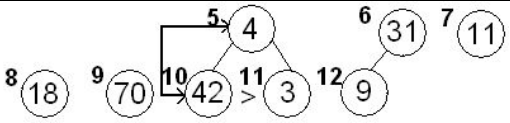
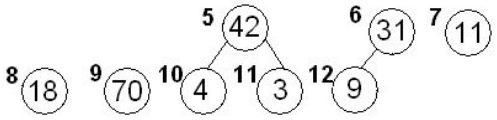
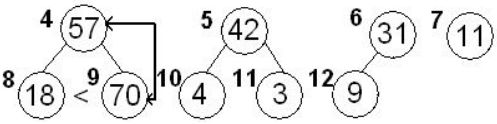
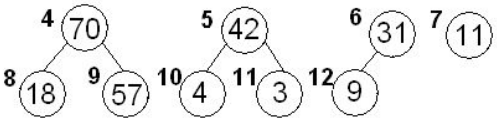
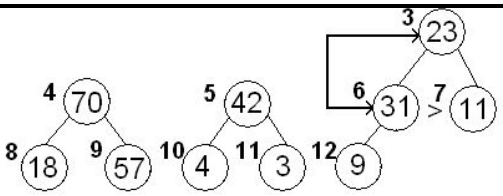
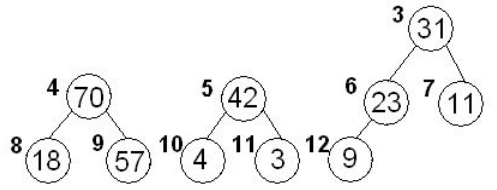
Решение

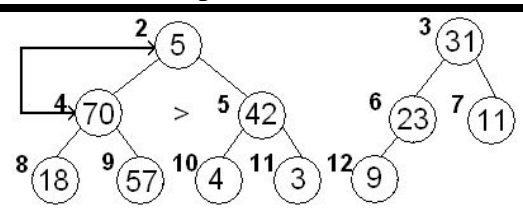
1 шаг



Последовательность 11,18,70,42, 3, 9 – пирамида

7 8 9 10 11 12

	Наращиваем пирамиду слева элементом 31 и просеиваем этот элемент по его потомкам: **31**,11,18,70,42, 3, 9 *6 7 8 9 10 11 12* Т.к. 31>9, оставляем 31 на своем месте. Полученная последовательность – пирамида.
	Наращиваем пирамиду слева элементом 4 и просеиваем этот элемент по его потомкам **4**,31,11,18,70,42, 3, 9 *5 6 7 8 9 10 11 12* 4<42 42>3
	Меняем местами элементы 4 и 42 42,31,11,18,70, **4**, 3, 9 *5 6 7 8 9 10 11 12* У элемента 4 потомков нет, просеивание закончено, полученная последовательность – пирамида.
	Наращиваем пирамиду слева элементом 57 и просеиваем этот элемент по его потомкам **57**,42,31,11,18,70, 4, 3, 9 *4 5 6 7 8 9 10 11 12* 57<70 18<70
	Меняем местами элементы 57 и 70 70,42,31,11,18,**57**, 4, 3, 9 *4 5 6 7 8 9 10 11 12* У элемента 57 потомков нет, просеивание закончено, полученная последовательность – пирамида.
	Наращиваем пирамиду слева элементом 23 и просеиваем этот элемент по его потомкам **23**,70,42,31,11,18,57, 4, 3, 9 *3 4 5 6 7 8 9 10 11 12* 23<31 31>11
	Меняем местами элементы 23 и 31 31,70,42,**23**,11,18,57, 4, 3, 9 *3 4 5 6 7 8 9 10 11 12* Т.к. 23>9, оставляем 23 на своем месте. Полученная последовательность – пирамида.



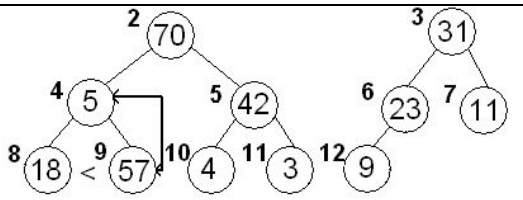
Наращиваем пирамиду слева элементом 5 и просеиваем этот элемент по его потомкам

5,31,70,42,23,11,18,57, 4, 3, 9

2 3 4 5 6 7 8 9 10 11 12

70>42

5<70



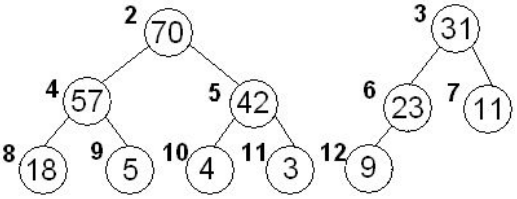
Меняем местами элементы 5 и 70 и просеиваем далее элемент 5 по его потомкам

70,31,**5**,42,23,11,18,57, 4, 3, 9

2 3 4 5 6 7 8 9 10 11 12

18<57

5<57

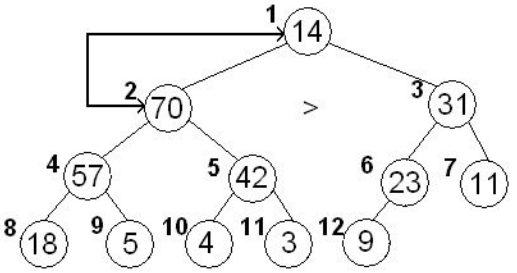


Меняем местами элементы 5 и 57

70,31,57,42,23,11,18, **5**, 4, 3, 9

2 3 4 5 6 7 8 9 10 11 12

У элемента 5 потомков нет, просеивание закончено, полученная последовательность – пирамида.



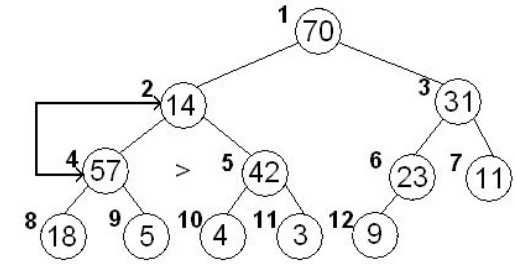
Наращиваем пирамиду слева элементом 14 и просеиваем этот элемент по его потомкам

14,70,31,57,42,23,11,18, 5, 4, 3, 9

1 2 3 4 5 6 7 8 9 10 11 12

70>31

14<70



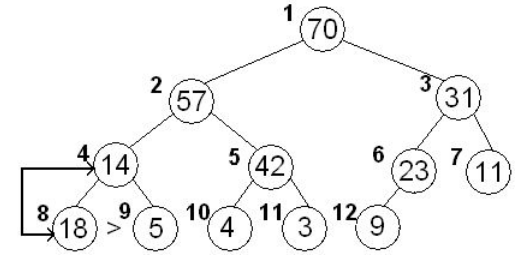
Меняем местами элементы 14 и 70 и просеиваем далее элемент 14 по его потомкам

70,**14**,31,57,42,23,11,18, 5, 4, 3, 9

1 2 3 4 5 6 7 8 9 10 11 12

57>42

14<57



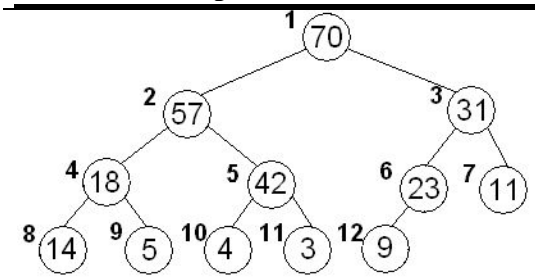
Меняем местами элементы 14 и 57 и просеиваем далее элемент 14 по его потомкам

70,57,31,**14**,42,23,11,18, 5, 4, 3, 9

1 2 3 4 5 6 7 8 9 10 11 12

18>5

14<18



Меняем местами элементы 14 и 18

70,57,31,18,42,23,11,**14**, 5, 4, 3, 9

1 2 3 4 5 6 7 8 9 10 11 12

У элемента 14 потомков нет, просеивание закончено, полученная последовательность – пирамида.

Выполнение шага 1 закончено, построенная последовательность, содержащая все элементы массива *Mass*, пирамида.

2 шаг

2.1. *Mass*=(70,57,31,18,42,23,11,14,5,4,3,9). Меняем местами в пирамиде первый и последний элементы 9,57,31,18,42,23,11,14, 5, 4, 3,70

1 2 3 4 5 6 7 8 9 10 11 12

Усекаем справа последовательность на 1 элемент (он стоит на своем месте)

9,57,31,18,42,23,11,14, 5, 4, 3 || **70**

1 2 3 4 5 6 7 8 9 10 11

2.2. Просеиваем элемент Mass(1)=9 по своим потомкам

9,57,31,18,42,23,11,14, 5, 4, 3

1 2 3 4 5 6 7 8 9 10 11

57>31

9<57

57, **9**,31,18,42,23,11,14, 5, 4, 3

1 2 3 4 5 6 7 8 9 10 11

18<42

9<42

57,42,31,18, **9**,23,11,14, 5, 4, 3

1 2 3 4 5 6 7 8 9 10 11

4>3

9>4

2.1 Полученная последовательность (57,42,31,18,9,23,11,14,5,4,3) – пирамида. Меняем местами первый и последний элементы последовательности и усекаем справа последовательность на один элемент (он стоит на своем месте в получаемом массиве)

3,42,31,18, 9,23,11,14, 5, 4 || 57

1 2 3 4 5 6 7 8 9 10

2.2 Просеиваем элемент Mass(1)=3 по его потомкам и получаем пирамиду

42,18,31,14,9,23,11,3,5,4

Шаги	Элементы рассматриваемой последовательности	Массив *Mass*
2.1	4,18,31,14,9,23,11,3,5 || 42	(4,18,31,14,9,23,11,3,5,||42,57,70)

2.2	31,18,23,14,9,4,11,3,5	(31,18,23,14,9,4,11,3,5,||42,57,70)
2.1	5,18,23,14,9,4,11,3 || 31	(5,18,23,14,9,4,11,3,||31,42,57,70)
2.2	23,18,11,14,9,4,5,3	(23,18,11,14,9,4,5,3,||31,42,57,70)
2.1	3,18,11,14,9,4,5 || 23	(3,18,11,14,9,4,5,||23,31,42,57,70)
2.2	18,14,11,3,9,4,5	(18,14,11,3,9,4,5,||23,31,42,57,70)
2.1	5,14,11,3,9,4 || 18	(5,14,11,3,9,4,||18,23,31,42,57,70)
2.2	14,9,11,3,5,4	(14,9,11,3,5,4,||18,23,31,42,57,70)
2.1	4,9,11,3,5 || 14	(4,9,11,3,5,||14,18,23,31,42,57,70)
2.2	11,9,4,3,5	(11,9,4,3,5,||14,18,23,31,42,57,70)
2.1	5,9,4,3 || 11	(5,9,4,3,||11,14,18,23,31,42,57,70)
2.2	9,5,4,3	(9,5,4,3,||11,14,18,23,31,42,57,70)
2.1	3,5,4 || 9	(3,5,4,||9,11,14,18,23,31,42,57,70)
2.2	5,3,4	(5,3,4,||9,11,14,18,23,31,42,57,70)
2.1	4,3 || 5	(4,3,||5,9,11,14,18,23,31,42,57,70)
2.2	4,3	(4,3,||5,9,11,14,18,23,31,42,57,70)
2.1	3 || 4	(3,||4,5,9,11,14,18,23,31,42,57,70)
2.2	3	(3,4,5,9,11,14,18,23,31,42,57,70)

Сортировка закончена. Отсортированный массив имеет вид:

Mass=(3,4,5,9,11,14,18,23,31,42,57,70)

Выбор наибольшего из (*n*-1) элемента массива, из (*n*-2) элементов массива и т.д. упрощается за счет получения большей информации при нахождении наибольшего из *n* элементов массива. Для больших *n* пирамидальная сортировка оказывается очень эффективной, она требует $O(n \cdot \log_2 n)$ шагов, где *n* – число элементов массива.

2. Проектная технология – средство реализации личностно-ориентированного обучения

Поиск новых подходов и форм организации учебной работы с учащимися диктуется стремлением современной школы к развитию личности и интеллекта школьника в такой степени, чтобы выпускник школы был способен не только самостоятельно находить и усваивать ранее сгенерированную и обработанную информацию, но и сам мог генерировать новые идеи. Одним из направлений поиска решения этой проблемы является деятельностный подход к обучению и, в частности, использование в обучении метода проектов. Содержание и методика курса «Информатика и ИКТ» нацелены на формирование творческих и исследовательских качеств молодого человека. Для этого в руках педагога и ученика есть замечательный инструмент – ком-

пьютер. На уроках информатики школьник как настоящий исследователь наблюдает объекты и их поведение – информационные процессы, на основе наблюдений выдвигает гипотезу, проверяет её, используя компьютерную модель. Появляется исследовательское направление курса, в котором ключевым словом является творчество. Метод проектов в курсе информатики с успехом используется на трех этапах обучения: пропедевтическом, базовом, профильном. Значительная часть учебного времени (половина лабораторных работ) отводится работе школьников по системе индивидуальных и коллективных проектов. Такая форма работы позволяет учителю увидеть и использовать индивидуальные способности каждого школьника, и главное, используя новые современные информационные технологии обучения, привить детям вкус к творчеству и исследовательской деятельности.

Под учебным *проектом* понимается обоснованная, спланированная и осознанная деятельность обучаемого (обучаемых-партнеров), направленная на формирование у него (у них) определенной системы интеллектуальных и практических умений.

Целью метода проектов является развитие самообразовательной активности у учащихся. В результате творческой практической деятельности обучаемые создают конечный продукт в виде новых знаний и умений.

Метод проектов был разработан в начале XX века американским философом и педагогом Дж. Дьюи и его учеником В.Х. Килпатриком с целью ориентирования обучения на целенаправленную деятельность детей с учетом их личных интересов. В России идеи проектного обучения возникли практически параллельно с работами американских педагогов. Русский педагог С.Т. Шацкий в 1905 году организовал небольшую группу сотрудников, пытавшуюся активно использовать проектные методы в практике преподавания.

Сформулируем некоторые методические рекомендации для организации учебной деятельности учащихся в форме работы над проектом.

1. Учитель посвящает один урок беседе о проекте (системе проектов), подробно рассказывает о теме (темах) проектов, формули-

рует общую цель проекта, приводит примеры реализации некоторых подцелей. Встречные предложения учащихся приветствуются, а самостоятельно выдвинутая и обоснованная тема или конкретизированная цель проекта заслуживает особенного учительского внимания и поощрения.

2. Учащимся представляется достаточно широкий набор проектов (широкий набор возможностей реализации одного проекта) для реализации возможности реального выбора. Проекты могут быть как индивидуальными, так и коллективными.

3. Проект должен быть посильным для ученика. Работа над проектом мотивируется интересом обучаемого, а не только ответственностью.

4. Проект должен побуждать к получению новых знаний. Получение новых знаний правильно мотивируется, и этот мотив выставляет не преподаватель, а сам ученик.

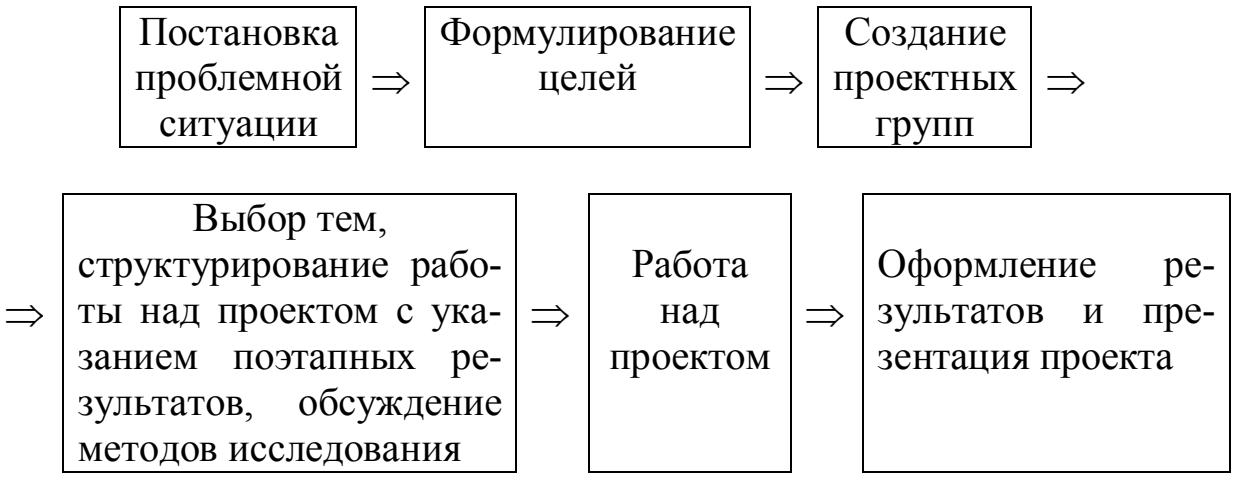
5. Для организации работы над проектом, обучаемый может быть снабжен инструкцией по работе над проектом.

6. При работе над проектом ученику необходим консультант; в самом начале работы особенно важен индивидуальный контакт с преподавателем. Целесообразно для консультационной работы приглашать школьников-старшеклассников. Если у ученика нет опыта самостоятельной творческой работы, возможно «внедрение» преподавателя в исследовательскую группу детей на принципе равных интересов. Если ученик видит интерес преподавателя к работе, к конечному результату, он заражается от него этим интересом. Если при этом преподаватель откажется от поучений, а будет просто более опытным товарищем, которому интересен как проект, так и само общение, то достижение цели проекта гарантировано. Появляется уникальная возможность управлять работой учебной исследовательской группы изнутри.

7. При работе над проектом необходимо создавать условия, при которых школьники имеют возможность обсуждать друг с другом свои успехи и неудачи, при этом происходит взаимообучение.

8. Важна практическая значимость полученного результата работы над проектом и оценка этого результата со стороны окружающих, общественное признание результата. Получение результата – мотив работы над проектом. Учебный проект должен предполагать для исполнителя законченность и целостность проделанной им работы. Необходимо, чтобы завершенный проект был презентован и получил внимание взрослых и сверстников.

Этапы деятельности над проектом можно изобразить в виде схемы:



Задание. Разработайте проект по теме «Алгоритмизация и программирование». Вам дано одно задание по этой теме, необходимо расширить задание:

- обогатите проект познавательными, занимательными фактами по теме, желательно связанными с предложенным заданием;

- если возможно, предложите общий метод решения представленной задачи;

- разбейте задачу на подзадачи, решение которых приведет к решению главной задачи;

- усложните задачу, придумайте продолжение предложенной задачи;

- приведите примеры интересных познавательных задач и их решения по теме «Алгоритм и его свойства. Способы описания алгоритма»;

- разработайте и реализуйте презентацию Вашего проекта.

Проект 1. *Суффиксная запись выражений*

Некоторые понятия

Константы и переменные назовем операндами. При обычной записи выражений, называемой *инфиксной*, символы двуместной операции пишутся между операндами, $k+m$. В математике используется *префиксная* запись, при которой символы операций записываются перед операндами, sin(b), arccos(c). Здесь символы sin, arccos есть имена функций, они располагаются перед операндами. Можно рассматривать все операции, как имена функций, которые могут быть помещены до операндов или после них, так вместо x-y можно записать –; x; y или x; y ;–. Способы записи выражений, в которых символ двуместной операции стоит до или после операндов, а не разделяет их, предложены польским логиком Яном Лукасевичем (Jan Lukasiewiez) (1878-1957) и принято называть их польской *префиксной* или польской *суффиксной* записями соответственно.

Примеры суффиксной записи выражений. Так как в суффиксной записи имена двух переменных или констант могут стоять непосредственно одно за другим, то для их разделения применяется символ (;), назовем его разделителем.

Обычная запись выражений	Суффиксная запись	Обычная запись выражений	Суффиксная запись
a/b	$a;b;/$	m-n	$m;n;-$
$k+l$	$k;l;+$	$\sqrt{a}$	$a;\sqrt{}$
$d*c$	$d;c;*$	sin(a)	a;sin

Вычисление значения выражения в суффиксной записи. Порядок выполнения операций в суффиксной записи определен порядком следования операндов и операций в выражении.

Алгоритм вычисления значения выражения можно описать так:

1. Просматриваем запись выражения слева направо.

2. Если встретится одноместная операция и предшествующий ей операнд, то применяем эту операцию к операнду и заменяем последовательность (операнд – операция) значением, т.е. операндом. Если встретится двуместная операция и предшествующие ей два операнда, то применяем операцию к операндам и заменяем последовательность (операнд – операнд – операция) значением, т.е. операндом

и переходим к шагу 1; иначе значение выражения найдено, процесс вычисления закончен.

Например, вычислить значение выражения $k;d;*;\sin;b;\sqrt{};+$ при заданных значениях переменных k, d, b можно так:

1. Просматривая запись выражения слева направо, заменяем последовательность $k;d;*$ значением выражения z_2, равным $k*d$, последовательность $b;\sqrt{}$ – значением выражения z_1, равным $\sqrt{b}$.

$$\underbrace{k;d;*}_{z_2};\sin;\underbrace{b;\sqrt{}}_{z_1};+$$

Получим запись выражения $z_2;\sin;z_1;+$.

2. Просматривая запись выражения, заменяем последовательность $z_2;\sin$ значением z_3, равным $\sin(z_2)$.

$$\underbrace{z_2;\sin;}_{z_3}z_1;+$$

Получим запись выражения $z_3;z_1;+$.

3. Заменяем последовательность $z_3;z_1;+$ значением z_4, равным z_3+z_1.

Пример. Вычислите значение выражения $5;3,1;2,9;+;-;3;\text{“}x^2\text{”};*$.

Решение:

1. $5;\underbrace{3,1;2,9;+}_{6=3,1+2,9};-;\underbrace{3;\text{“}x^2\text{”}}_{9=3^2};*$

2. $\underbrace{5;6;-}_{-1=5-6};9;*$

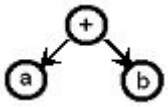
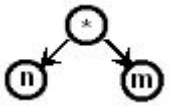
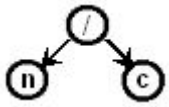
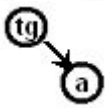
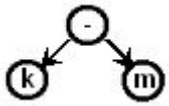
3. $\underbrace{-1;9;*}_{-9=-1*9}$

-9 – значение данного выражения.

Преобразование обычной записи выражений в суффиксную. Для преобразования обычной записи выражений в суффиксную запись изобразим выражение в виде бинарного дерева. Дерево – совокупность конечного числа узлов – вершин и попарно соединяющих эти вершины линий, называемых дугами или ребрами дерева. Первоначально выбранная вершина называется корнем дерева. Каждая операция соответствует вершине с меткой (символ операции). От вершины с символом операции исходят дуги, ведущие к операндам. В бинар-

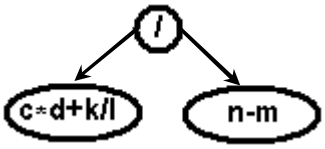
ном дереве каждый узел имеет самое большее два поддерева; в случае, когда имеется два поддерева, мы различаем левое и правое поддерево. Первый операнд двуместной операции мы изображаем в левом поддереве, второй операнд в правом поддереве.

Рассмотрим примеры представления выражения в виде бинарного дерева.

Выражение	Дерево выражения	Выражение	Дерево выражения
a+*b*		*n**m	
n/*c*		tg(*a*)	
k-*m*			

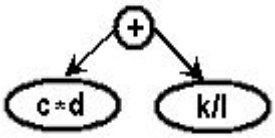
Пример. Изобразите выражение (*c***d*+*k*/*l*)/(*n*-*m*) в виде бинарного дерева.

Решение. Первая вершина дерева – корень, соответствует последней операции (делению) при вычислении значения этого выражения.

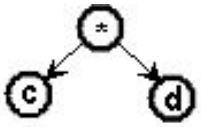


Первый операнд (*c***d*+*k*/*l*) изобразим в левом поддереве. Второй операнд *n*-*m* – в правом поддереве.

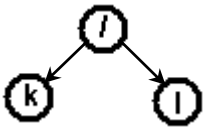
Операнд (*c***d*+*k*/*l*) можно изобразить как бинарное дерево:



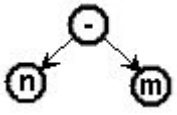
Операнд *c***d* – как дерево:



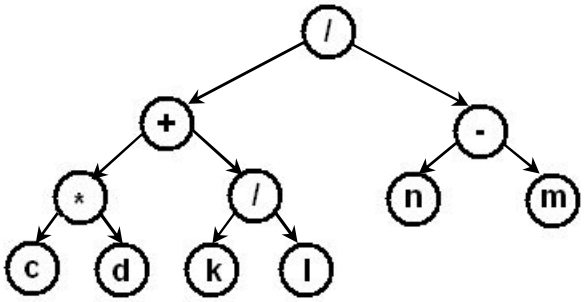
Операнд *k*/*l* – как дерево:



Операнд *n*-*m* – как дерево:



Подставляя детализированные узлы, получаем дерево данного выражения:



Для работы с древовидными структурами имеется множество алгоритмов с одной идеей – *прохождением дерева*. Это способ последовательного исследования узлов дерева, при котором каждый узел проходится точно один раз. Полное прохождение дерева дает линейное упорядочивание узлов. Для прохождения дерева используем концевой порядок прохождения: пройти левое поддерево, пройти правое поддерево, попасть в корень. При прохождении дерева выписываем символы в вершинах дерева, получаем линейную запись меток вершин – это и есть суффиксная запись выражения. В нашем примере это: *c*;*d*;∗;*k*;*l*;/;+;*n*;*m*;-;/.

Обратная польская запись весьма полезна при вычислении выражений на компьютере.

Задание 1 к проекту 1. Напишите алгоритм для бездумного исполнителя преобразования записи обычного алгебраического выражения в обратную польскую (суффиксную) форму записи, т.е. *A*+(*B*-*C*)∗*D*-*F*/(*G*+*H*) в форму записи – *A*;*B*;*C*;-;*D*;∗;+;*F*;*G*;*H*;+;/;-.

Задание 2 к проекту 1. Задано уравнение вида f(*x*)=0, где выражение f(*x*) состоит из целых чисел, арифметических операций +,-,*,/ и переменной *x*, которая может входить в выражение не более одного раза. Выражение задаётся в обратной польской записи. Написать программу, которая решает заданное уравнение f(*x*)=0 и печатает все его корни. Длина исходной строки не превышает 80 символов. Элементы обратной польской записи разделяются пробелами.

Технические требования:

Программа должна обеспечивать следующий режим работы:

- печатает приглашение «>» и ожидает ввода строки;

- рассматривая введённую строку как обратную польскую запись выражения f(x), определяет и печатает все корни уравнения f(x) = 0, если корни отсутствуют, то печатает сообщение «корней нет».

Пример:

> 2 *X* * 4 - 7 / , ответ – уравнение имеет единственный корень 2.

Проект 2. *Лабиринт.* Попав в лабиринт, состоящий из одинаковых комнат, каждая из которых может иметь от 1 до 4 дверей в соседние комнаты, путник долго блуждая по нему, прошёл все имеющиеся в этом лабиринте двери (возможно не по одному разу), пока не нашел выход. На всякий случай путник составил описание своего маршрута, обозначая в каждой комнате направления движения соответственно буквами С (север), В (восток), Ю (юг), З (запад). Напишите алгоритм, который по заданному описанию маршрута путника определяет:

а) какой-нибудь другой допустимый путь;

б) какой-нибудь допустимый путь без захода дважды в одну и ту же комнату;

в) кратчайший из допустимых путей (длиной пути считается количество проходов через двери).

Проект 3. *Тетраэдр.* Одна из граней модели правильного тетраэдра красного цвета, остальные – синего. Модель поставили красной гранью на стол и стали «перекатывать» по его поверхности (перекатывая через ребро без скольжения). После нескольких таких переворачиваний модель вернулась на исходное место. Существует ли алгоритм «перекатывания» модели, пользуясь которым можно вернуть модель на исходное место так, чтобы она стояла на столе одной из синих граней. Если существует, то напишите этот алгоритм, если нет, то докажите это утверждение.

Проект 4. *Максимальное значение*. Дано целое десятичное число N ($1 \le N \le 65535$). Некто записал это число в двоичном формате и стал циклически сдвигать вправо, т.е. брать последнюю цифру числа и переносить ее в начало. Например, если N=11, то в двоичном формате оно будет представлено как 1011. После первого сдвига получится 1101, после второго – 1110, после третьего – 0111, после четвертого – исходное число 1011. Легко видеть, что максимальное значение из всех полученных таким образом чисел будет иметь число

1110, и это значение равно 14. Напишите алгоритм, который для заданного числа *N* определяет максимальное число из чисел, которые могут получаться в результате вышеописанных сдвигов.

Проект 5. *Перестановки.* Перестановкой из *n* различных элементов называется последовательность этих элементов, записанная, может быть, в другом порядке. Перестановки *n* чисел (1, 2, … , *n*) можно упорядочить, считая, что та из двух перестановок больше, у которой больше первый элемент. При одинаковых первых элементах больше та, у которой больше второй элемент и т.д. Например, перестановка (3, 5, 6, 1, 4, 2) больше, чем (3, 5, 4, 6, 2, 1). Перестановка (1, 2, 3, 4, 5, 6) – самая маленькая. Напишите алгоритм, определяющий по заданной перестановке, непосредственно следующую за ней, т.е. перестановку большую, чем заданная, но меньшую, чем любая другая, большая, чем заданная. Входные данные: число *n* – количество чисел в перестановке и последовательность из *n* чисел, образующих эту перестановку.

Проект 6. *Квадрат палиндрома.* Число называется палиндромом, если его запись читается одинаково с начала и с конца. Напишите алгоритм нахождения всех шестизначных палиндромов, квадраты которых также являются палиндромами. Из числовых типов разрешается использовать только тип Integer.

Проект 7. *Латинские квадраты.* Матрица *A* называется *латинским квадратом* порядка *N*, если каждое из чисел 1, 2, … , *N* входит *ровно один раз в каждую строку* и *в каждый столбец* матрицы *A*. Напишите алгоритм вычисления количества различных латинских квадратов с фиксированной первой строкой 1, 2, … , *N* для $1<N<7$.

Из истории латинских квадратов

Считается, что название «латинский квадрат» для обозначения описанной матрицы связано с тем, что Леонард Эйлер около 1780 г. использовал латинские буквы для обозначения символов внутри квадрата. Заинтересовавшись старой математической головоломкой, он установил существование пар ортогональных латинских квадратов любого порядка, не имеющего вид $4n+2$, но, несмотря на длительные поиски, не смог найти ни одной пары шестого порядка. В конце концов, он предположил, что не существует пар латинских квадратов шестого порядка и любого порядка вида $4n+2$.

Гипотеза Эйлера продержалась столетия; для шестого порядка она была доказана только в начале двадцатого века путем исчерпывающего перебора всех вариантов, что без применения ЭВМ представляет собой задачу необычайной трудности. Когда стали доступны ЭВМ, их можно было бы использовать для решения задачи десятого порядка, но в данном случае они появились на сцене слишком поздно, так как в 1959 г. гипотеза была, наконец, опровергнута за счет общего построения, приводящего (без ЭВМ) к ортогональным латинским квадратам любого порядка $4n+2 \geq 10$. Впоследствии были составлены сравнительно быстрые (учитывая сложность этой комбинаторной задачи) программы для нахождения многих пар десятого порядка.

Многие задачи, связанные с очередностью событий, составлением расписания, распределением заданий или ресурсов при их идеализации (чтобы задача стала более простой) можно сформулировать при помощи латинских квадратов. Приведем пример такой задачи. В школе четыре преподавателя T_1, T_2, T_3, T_4 в течение дня проводят занятия с четырьмя классами C_1, C_2, C_3, C_4. Каждый преподаватель должен дать урок в каждом классе один раз в течение дня. Написать возможные варианты составления расписания на один день.

$A=$

	C_1	C_2	C_3	C_4
T_1	1	2	3	4
T_2	2	3	4	1
T_3	3	4	1	2
T_4	4	1	2	3

Возможное расписание можно изобразить матрицей A, число K в строке i и столбце j означает, что преподаватель T_i занимается в классе C_j во время K.

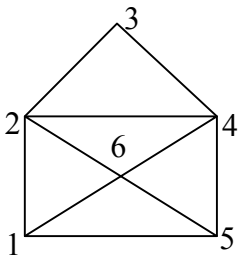
Легко видеть, что данное расписание – латинский квадрат, так как один преподаватель не может одновременно заниматься с двумя классами и никакое целое число (час) не может встретиться дважды в одной строке; так как два преподавателя не могут одновременно заниматься с одним и тем же классом. Никакое целое число (час) не встречается дважды в одном столбце.

Проект 8. *Ортогональные латинские квадраты.* Матрица A размером $N \times N$ назовем *латинским* квадратом (см. Проект 7) порядка N, если каждое из чисел 1, 2, … , N входит ровно один раз в каждую строку и в каждый столбец этой матрицы. Латинские квадраты A и B назовем *ортогональными***,** если при их наложении в новой матрице получается множество всех упорядоченных пар элементов (среди N пар соответствующих компонентов нет двух одинаковых). Например, для ниже приведенных латинских квадратов A и B матрица C определяет их ортогональность.

$$A=\begin{vmatrix}1&2&3\\2&3&1\\3&1&2\end{vmatrix},\ B=\begin{vmatrix}1&2&3\\3&1&2\\2&3&1\end{vmatrix},\ C=\begin{vmatrix}(1,1)&(2,2)&(3,3)\\(2,3)&(3,1)&(1,2)\\(3,2)&(1,3)&(2,1)\end{vmatrix}$$

Напишите алгоритм, в результате исполнения которого будут получены по одной паре ортогональных латинских квадратов для всех значений N от 2 до 9 в формате матрицы C.

Проект 9. *Уникурсальная линия.* Линия называется уникурсальной, если её можно начертить, не отрывая карандаша от бумаги и не проходя два раза одно и то же звено. Линии, содержащей N узлов, можно сопоставить квадратную матрицу порядка N – матрицу смежности, элемент a(i,j), которой равен 1, если узел i соединен с узлом j некоторым отрезком, не содержащим других узлов, и 0 в противном случае (i, j=1, 2, … , N). Для линии, изображенной на рисунке, матрица смежности имеет вид:



	1	2	3	4	5	6
1	0	1	0	0	1	1
2	1	0	1	1	0	1
3	0	1	0	1	0	0
4	0	1	1	0	1	1
5	1	0	0	1	0	1
6	1	1	0	1	1	0

Дана матрица соединений с N узлами. Напишите алгоритм, определяющий является ли линия, заданная матрицей смежности, уникурсальной, если да, то алгоритм должен находить хотя бы одну последовательность номеров узлов, которые образуют требуемый путь.

Замечание. Узлы A и B смежные, если существует отрезок, соединяющий узел A с узлом B. Линия называется связной, если все ее узлы связаны. Известно, что линия уникурсальна тогда и только тогда, когда она связна и число тех ее узлов, из которых выходит нечетное число звеньев, не больше двух.

Проект 10. *Транспортная сеть.* В городе имеется M площадей и K дорог, соединяющих эти площади. Каждая дорога соединяет две площади, длина каждой дороги равна 1.

Задания.

1. Требуется спланировать одностороннее движение по дорогам так, чтобы с любой площади можно было бы проехать до любой другой.

2. Решить задачу так, чтобы максимальное время движения между двумя площадями было минимальным.

Образец ввода: Введите количество площадей: *M* Введите количество дорог *K* Дорогой 1 связаны площади: …….. Дорогой *k* связаны площади: ……	*Замечание.* При выводе результата для каждой дороги указать площади в порядке, соответствующем направлению спланированного движения. Для второй задачи необходимо указать максимальное время движения между парами площадей, если время движения между двумя любыми площадями, соединёнными дорогой, равно 1. Количество площадей не превосходит 50.

Проект 11. *Переправа.* На берегу реки находятся: лодочник, волк, коза и капуста. У берега реки расположена лодка. Лодочник должен переправить волка, козу и капусту на другой берег реки. При перевозке в лодке с лодочником может находиться либо только волк, либо только коза, либо только капуста. Если оставить волка с козой или козу с капустой без лодочника, то волк съест козу или коза съест капусту. Разработать программу игрового характера, моделирующую действия лодочника.

Технические требования	
Программа должна изображать начальное состояние переправы, реализовывать управление выбором переправляющихся, изображать состояние переправы в результате этого выбора.	В программе так же нужно реализовать управление ходом переправы, перемещение переправляющихся и изображение состояния переправы после каждого такта перемещения.

3. Пример использования элементов технологии проблемного обучения при введении команды повторения «пока», управляющей команды организации действий в алгоритмах

> «Психическое развитие, особенно интеллектуальное развитие человека, осуществляется только в условиях преодоления "препятствий", интеллектуальных трудностей. Нужда, потребность – главный источник психического развития человека.
>
> *Матюшкин А.М.*

Необходимые знания по информатике, умения и навыки учащиеся приобретают только путём самостоятельных интеллектуальных усилий. Учитель должен направлять учащихся, организовывать учебный процесс, используя различные формы, методы и средства обучения так, чтобы учение стало интересным и продуктивным. Команда «повторение с предусловием» является одной из тем содержательной линии «Алгоритмизация и программирование». При изучении этой темы желательно использовать частично-поисковый метод проблемного обучения, эвристический метод. *Проблемное обучение* – система методов и средств, обеспечивающих возможности творческого участия учащихся в процессе усвоения новых знаний, формирование творческого мышления и познавательных интересов личности. Этот метод в информатике предполагает мотивацию введения новых команд при изучении алгоритмических языков, реализацию процесса «переоткрытия» этой команды, наподобие процесса её открытия разработчиками изучаемой системы программирования. При этом учитель подводит учащихся к «переоткрытию» новой команды, расчленяет этот процесс на этапы, облегчая самостоятельную и творческую деятельность учащихся, сокращает время на решение проблемной задачи.

Изучение нового материала можно провести по следующему плану:

1. Проверка домашнего задания и подготовка к изучению нового материала.

2. Постановка проблемной задачи. Эвристическая беседа, при проведении которой учащиеся подводятся к необходимости введения новой команды языка программирования для описания алгоритмов решения практических задач на компьютере.

3. «Переоткрытие» учащимися команды «повторения с предусловием», формирование синтаксиса команды, алгоритма её выполнения.

4. Решение проблемной задачи с использованием введенной команды, описание условий, когда желательно использовать команду «повторение с предусловием», выделение преимуществ использования этой команды при описании алгоритмов решения практических задач.

5. Использование команды «повторения с предусловием» при решении практических задач.

Детализируем некоторые этапы урока. Учащиеся до того изучили тему «Графические возможности языка программирования Turbo Pascal».

1 этап. На предыдущем уроке учащимся можно дать опережающее задание.

Задача 1. Описать алгоритм рисования на экране 5 концентрических окружностей с центром в точке (300,250) и с радиусом *r*. Для первой окружности *r* равно 20, для второй – *r* равно 40, … , для пятой окружности *r* равно 180.

Задача 2. Три туриста подошли к реке, по которой катаются на лодке два мальчика. Описать алгоритм переправы на другой берег туристов, если лодка вмещает либо одного туриста, либо одного или двух мальчиков, а мальчика и туриста уже не вмещает.

При анализе и сравнении алгоритмов решения поставленных задач *учащиеся отмечают*, что алгоритмы содержат многократно повторяющиеся команды: 5 раз повторяются две команды в первом алгоритме, 3 раза повторяется серия из 4-х команд во втором алгоритме.

Предполагаемые решения задач.

1. Алгоритм построения окружностей.

```
program circle1;
uses crt,graph;
var gd,gm,r,c:integer;
begin
   clrscr;
   gd:=detect; gm:=detect;
   initgraph (gd, gm,'');
   r:=0;
 r:=r + 20;
 circle(300,250,r);
 r:=r+20;
 circle(300,250,r);
 r:=r+20;
 circle(300,250,r);
 r:=r+20;
 circle(300,250,r);
 r:=r+20;
 circle(300,250,r);
   closegraph;
end.
```

2. Алгоритм переправы туристов.

```
алг переправа
нач
Два мальчика на левый берег;
Один мальчик на правый берег;
Один турист на левый берег;
Один мальчик на правый берег;
Два мальчика на левый берег;
Один мальчик на правый берег;
Один турист на левый берег;
Один мальчик на правый берег;
Два мальчика на левый берег;
Один мальчик на правый берег;
Один турист на левый берег;
Один мальчик на правый берег;
кон
```

2 этап. Далее учащимся можно предложить проблемную задачу.

Задача 3. Описать алгоритм рисования на экране монитора 20 концентрических окружностей с радиусами, изменяющимися от 20 до 380 с шагом 20, и центром в точке (300,250).

В результате анализа решения поставленной задачи, *учащиеся приходят к необходимости* расширения системы команд исполнителя для компактной записи алгоритма, т. е. введения новой команды.

3 этап. Для формирования команды «повторение с предусловием» учащимся можно предложить *житейскую* задачу.

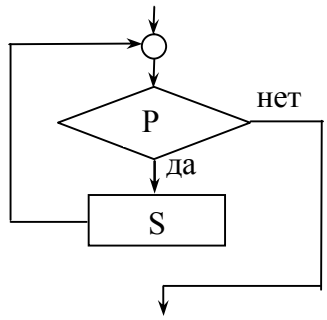
Задача 4. Описать алгоритм для исполнителя Перевозчика переправы с правого берега на левый берег туристов, прибывающих на правый берег в автобусе. Перевозчик не вникает в условие задачи, а умеет только формально командовать, отдавая приказания мальчикам и туристам. Условия перевозки как в задаче 2. Алгоритм должен быть выполнен без нашего участия.

Учащиеся предлагают сформировать для перевозчика такую команду: «пока на левом берегу есть хотя бы один турист, выполняй 4 команды, выделенные в задаче 2». Команда повторения сформирована, можно записать эту команду сначала на частично формализованном языке, затем на языке Turbo Pascal.

Частично формализованный язык	Язык программирования Turbo Pascal

нц пока *на левом берегу есть хотя бы один турист,* делай
Два мальчика на левый берег
Один мальчик на правый берег
Один турист на левый берег
Один мальчик на правый берег
кц
	r:= 20
	while r<380 do
	begin
	circle(300,250,r);
	r:=r+20;
	end;

Учащиеся формируют алгоритм исполнения команды с предусловием, так как они хотят, чтобы команда выполнялась, что совпадает с алгоритмом исполнения этой команды в языке Turbo Pascal.

Блок – схема исполнения команды «пока»	*Алгоритм исполнения команды повторения «пока»*
	По этой команде сначала проверяется условие *P*. Если оно выполняется, то производятся действия, определяемые функциональным блоком *S*, затем снова проверяется условие *P* и так далее. При невыполнении условия *P* (по стрелке с надписью «нет») происходит выход из цикла, далее выполняются операторы, следующие за данной командой.

Сформированная команда используется для компактной записи алгоритмов решения задач 3 и 4.

Переправа	Программа построения окружностей

алг переправа
нач
нц пока *на левом берегу есть хотя бы один турист,* делай
Два мальчика на левый берег
Один мальчик на правый берег
Один турист на левый берег
Один мальчик на правый берег
кц
кон
	program circlt2;
	uses crt,graph; var gd,gm,r,c:integer;
	begin
	clrscr; gd:=detect; gm:=detect;
	initgraph(gd,gm); r:=20;
	while r<380 do
	begin
	circle(300,250,r); r:=r+20;
	end;
	closegraph;
	end.

Учащиеся делают вывод, что использование команды повторения с предусловием делает решение задачи компактным. Команда повторения с предусловием задает небольшим описанием большое количество действий. Команды повторения в описании алгоритмов решения задач применяются тогда, когда необходимо повторить серию команд с изменяющимися параметрами многократно в одном и том же месте алгоритма.

Кодирование команды повторения «пока» на алгоритмических языках

<table>
<tr><td colspan="2">Turbo Pascal</td><td rowspan="2">QBasic</td><td rowspan="2">Ершол</td></tr>
<tr><td>Серия S состоит из одной команды</td><td>Серия S содержит более одной команды (R1,…, RK).</td></tr>
<tr><td colspan="4">Команда повторения «пока» (цикл с предусловием)</td></tr>
<tr><td>While P do
S;</td><td>While P do
begin
R1; R2;
….
RK
end;</td><td>Do while P
S
Loop</td><td>нц пока P
S
кц</td></tr>
</table>

4. Вопросы и задания к семинарским занятиям

Тема: Алгоритмизация и программирование

Выполните задания 1-9, результаты своей работы доложите в студенческой аудитории.

1. Определите место и роль темы «Алгоритмизация и программирование» в решении общеобразовательных задач предмета «Информатика и ИКТ». Какое место этой теме отводят авторы учебных пособий и почему? Какое место отвели бы Вы? Выбор обоснуйте. Как менялось со временем место и содержание темы «Алгоритмизация и программирование» в предмете «Информатика и ИКТ»?

2. Проведите сравнительный дидактический анализ содержания учебного материала по данной теме в различных учебниках и учебных пособиях для общеобразовательных учреждений на каждой ступени непрерывного курса изучения информатики. Соотнесите содержание учебного материала с требованиями государственного стандарта образования по информатике.

3. Сформулируйте цели и задачи, стоящие перед учителем в процессе организации изучения школьниками данной темы. Какие учебные цели соответственно должны стоять перед учащимися?

4. Проанализируйте программное обеспечение в поддержку изучения темы «Алгоритмизация и программирование». Определите дидактические цели использования выбранного программного обеспечения при изучении темы.

5. Выявите базовые понятия темы «Алгоритмизация и программирование», определите этапы, формы и методы их формирования, установите отношения между выделенными понятиями. Составьте терминологический словарь по базовым понятиям темы. Определите общеобразовательный и мировоззренческий аспекты базовых понятий темы.

6. Отберите содержание учебного материала по теме «Алгоритмы, свойства алгоритмов. Способы записи алгоритмов» в соответствии с уровнем психического развития школьника на выбранном конкретном возрастном этапе. Составьте логико-структурную модель отобранного учебного материала по теме.

7. Рассмотрите системы учебных исполнителей и их назначение в различных учебниках по информатике и ИКТ. Рассмотрите схемы знакомства с выбранными исполнителями (среда, система команд исполнителей). Какие основные положения составляют методику структурного подхода к алгоритмизации и программированию? Каким требованиям должен удовлетворять учебный исполнитель для использования его в учебном процессе по этой методике?

8. По каким критериям Вы хотите построить последовательность рассматриваемых на занятиях задач при изучении темы «Алгоритмы, свойства алгоритмов. Способы записи алгоритмов»? Приведите примеры задач, которые нужно рассмотреть с учащимися, для наиболее полного осознания ими понятия алгоритма. Представьте систему задач для формирования одного из основного понятий содержательной линии «Алгоритмизация и программирование». Мотивируйте свои предложения.

9. Составьте словесное, графическое описание игровых алгоритмов:

а) игра «Баше»; б) игра «Ханойская башня»;

в) игра «Жизнь»; г) другие игровые алгоритмы.

Продумайте возможность их использования при введении основных понятий данной темы.

10. Подготовьте конспекты уроков по предложенным ниже темам, учитывая направленность урока (урок по изучению нового материала; урок по закреплению изученного материала; урок проверки знаний, умений и навыков; урок по систематизации и обобщению изученного и др.):

а) алгоритмы и их свойства;

б) исполнители алгоритмов, схема знакомств с исполнителем;

в) способы записи алгоритмов;

г) формальное исполнение алгоритма, возможность автоматизации исполнения алгоритма;

д) базовые управляющие структуры организации действий в алгоритмах решения задач;

д) введение типов данных выбранного вами языка программирования[37];

е) выработка умений и навыков по составлению алгоритмов отобранных Вами задач на данном этапе изучения информатики.

Апробируйте проведение фрагментов разработанных уроков в студенческой аудитории.

11. Разработайте по выбранной схеме технологическую карту серии уроков по одной теме содержательной линии «Алгоритмизация и программирование». *Проведите презентацию своих разработок в студенческой аудитории.*

12. Проанализируйте дидактические возможности учебного материала содержательной линии «Алгоритмизация и программирование» на каждой ступени непрерывного курса изучения информатики для реализации задач:

[37] Предлагаемые языки программирования: Ершол, QBasic, Turbo Pascal, Visual Basic.net, Turbo Delphi, C# и др.

а) формирования системно-информационных представлений и информационной культуры учащихся в процессе изучения предмета «Информатика и ИКТ»

б) развивающего обучения.

Доложите результаты своего дидактического анализа в студенческой аудитории.

5. Лабораторно-практические работы

Изучите учебную, методическую, специальную литературу по предложенным ниже темам лабораторных работ. Составьте аннотированный список литературы. Выделите основные понятия предложенных тем. Разработайте методику и подберите совокупность вариативных заданий для формирования знаний, умений и навыков у учащихся по предложенным ниже темам. Разработайте методику обучения технологическим приемам решения задач по данным темам. Выполненную работу оформите и подготовьте для защиты.

Лабораторная работа 1

1. Система программирования «КУМИР». Работа в текстовом редакторе системы Кумир (написание текста программы, редактирование).

2. Учебные исполнители Робот, Чертёжник (Среда, СКИ). Непосредственное управление Роботом с использованием Пульта управления. Работа с редактором лабиринта. Линейные программы для рассмотренных исполнителей. Отладка программ, различные способы исполнения программ.

3. Вспомогательный алгоритм на пропедевтическом уровне.

4. Две технологии проектирования алгоритмов: «снизу вверх» и «сверху вниз».

Предлагаемые задания: стр. 64-84 [13]; стр. 145-154 глава 3; [5].

Лабораторная работа 2

1. Команды «обратной связи». Диалог «ЭВМ – Робот» при выполнении команд «обратной связи».

2. Схемы управления исполнителем. Методическая значимость рассмотрения схемы управления исполнителем с использованием промежуточного звена (компьютера) для мотивации введения управляющих команд организации действий в алгоритмах.

3. Управляющие команды организации действий в алгоритмах решения практических задач:

3.1. Команды повторения – *«цикл с предусловием», «цикл с постусловием»*.

3.2. Команда ветвления в полной и сокращенной формах.

3.3. Команда выбора.

Предлагаемые задания: стр. 86-113 [13]; стр. 35-45 глава 1; [5].

4. Правильный алгоритм. Примеры доказательств правильности алгоритмов. Тестирование алгоритмов. Примеры наборов тестов для рассматриваемых задач.

Предлагаемые задания: стр. 114-125 [13].

Лабораторная работа 3

1. Примеры алгоритмов с аргументами.
2. Команда присваивания в алгоритмах решения практических задач.
3. Логические величины в школьном алгоритмическом языке.
4. Алгоритмы с результатами.
5. Команды ввода/вывода.

Предлагаемые задания: стр. 136-180 [13].

Лабораторная работа 4

1. Организация данных в виде таблиц. Задачи, решения которых подводят учащихся к необходимости введения организации данных в виде таблиц (массивов).

2. Классификация задач, решение которых требует организации данных в виде таблиц. Например, по методам их решения, по основному содержанию задач и т. д.

3. Алгоритмы и программы на выбранном языке программирования поиска заданного элемента в таблице:

3.1. Поиск элемента в произвольной таблице.

3.2. Поиск элемента в отсортированной таблице.

4. Алгоритмы и программы на выбранном языке программирования сортировки элементов таблицы:

4.1 Простые сортировки: выбор, включение, обмен.

4.2 Пирамидальная сортировка.

4.3 Сортировка разделением.

4.4 «Внешняя» сортировка.

Предлагаемые задания: стр. 186-201 [13]; стр. 45-66 глава 1.

Лабораторная работа 5

1. Эвристическая беседа, опережающие задания для мотивации включения в учебный материал предмета «Информатика и ИКТ» темы «Символьные и литерные величины в языках программирования».

2. Криптография, некоторые методы сжатия информации.

3. Задания для формирования основных понятий данной темы:

- *Понятия* символьной и литерной величины в алгоритмических языках программирования Ершол, QBasic, Turbo Pascal.
- *Операции и функции,* определенные для *работы с символьными* величинами в алгоритмических языках программирования Ершол, Turbo Pascal.
- *Операции и функции*, определенные для *работы с литерными* величинами в системах программирования Ершол, QBasic, Turbo Pascal.

Предлагаемые задания: стр. 201-210 [13]; стр. 3-15 [19]; стр. 3-32 [20].

Программирование в среде ЛогоМиры

Лабораторная работа 6

1. Интерфейс системы ЛогоМиры, объекты среды программирования. Проекты, при работе с которыми учащиеся познакомятся со средой ЛогоМиры.

2. Команды языка Лого для управления Черепашкой, правила записи команд, параметры команды. Вычисления в среде ЛогоМиры. Информационная модель среды ЛогоМиры. Задания для формирования основных понятий изучаемой темы.

Предлагаемые задания: стр. 245-254 [9].

Лабораторная работа 7

1. Правила оформления программы в среде ЛогоМиры. Работа с программами, записанными на листе программ.
2. Линейные алгоритмы на языке Лого.
3. Разветвляющиеся алгоритмы на языке Лого.
4. Циклические алгоритмы на языке Лого.

Предлагаемые задания: стр. 255-274 [9].

Лабораторная работа 8

1. Понятия «процедура» и «модуль». Этапы решения сложных задач.
2. Переменная в алгоритме, имя и значение переменной в языке Лого.
3. Процедуры с параметрами. Правила создания и вызова процедур с параметрами.
4. Рекурсивные алгоритмы. Управляемая рекурсия.

Предлагаемые задания: стр. 275-289 [9].

Лабораторная работа 9

1. Списки в языке Лого.
2. Функции языка Лого для работы со списками.

Предлагаемые задания: стр. 345-362 [12].

Лабораторная работа 10

1. Внедрение музыки и звука в системе ЛогоМиры.
2. Синхронизация процессов в системе ЛогоМиры.
3. Запись звука в системе ЛогоМиры.
4. Работа с несколькими черепашками. Имена черепашек, активные черепашки. Обращение к черепашкам.
5. Создание ролевых мультфильмов в системе ЛогоМиры.

Предлагаемые задания: стр. 298-306 [9].

Моделирование знаний и логическое программирование

Лабораторная работа 11

1. Искусственный интеллект. История развития. Два подхода к реализации ИИ. Основные направления развития ИИ.
2. Представление знаний в системах искусственного интеллекта. Модели представления знаний. Машина вывода.

3. Экспертные системы. Схема функционирования ЭС. Классификация ЭС. Примеры существующих ЭС.

4. Логические основы языка Пролог. Основные объекты языка Пролог. Факт, правило, цель (вопрос) на языке Пролог. Логический вывод.

5. Решение логических задач.

Предлагаемые задания: стр. 258-264 [8].

6. Арифметические операции на языке Пролог. Рекурсия.

Предлагаемые задания: стр. 264-271 [8].

7. Разработка простых баз знаний на языке Пролог.

Предлагаемые задания: стр. 251-258, стр.268-269 [8].

8. Списки. Операции над списками.

Задание. Составьте базы знаний для решения следующих задач:

a) Проверки принадлежности элемента X списку L (belong(X,L)).

b) Определения длины списка L (len(L,N)).

c) Определения номера элемента X в списке L (number(X,L,N)).

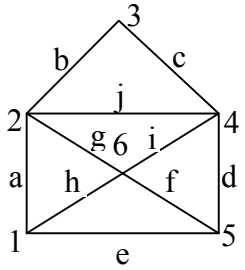
d) Объединения двух списков L1, L2 в один список L3 (Concat(L1,L2,L3)).

e) Нахождения максимального элемента X в списке L (max(L,X)).

f) Определения элемента X по его номеру N в списке L (ord(X,L,N)).

g) Сортировки списка L, используя метод сортировки простыми включениями.

h) Напишите программу на языке Пролог, определяющую является ли заданная линия уникурсальной. Если да, то выведите последовательность рёбер рисования фигуры не отрывая карандаша от бумаги и не проходя два раза по одной и той же линии.

	ребро(a,1,2) ребро(b,2,3) ребро(c,3,4) ребро(d,4,5) ребро(e,5,1) ребро(h,1,6)	ребро(a,2,1) ребро(b,3,2) ребро(c,4,3) ребро(d,5,4) ребро(e,1,5) ребро(h,6,1)

	ребро(i,6,4) ребро(f,5,6) ребро(g,6,2) ребро(j,2,4)	ребро(i,4,6) ребро(f,6,5) ребро(g,2,6) ребро(j,4,2)

6. Самостоятельная работа

При выполнении заданий 1-3 разработайте и реализуйте с использованием выбранного программного обеспечения учебные проекты в виде компьютерных моделей. *Подготовьте проекты к защите.*

1. Разработайте собственного исполнителя программ, с помощью которого можно формировать у учащихся базовые понятия темы «Алгоритмизация и программирование».

2. Выполните задание на стр. 194 для двух предложенных в работе или выбранных вами тем.

3. Разработайте сценарии программ (демонстрирующих, обучающих, тренажеров, контролирующих и др.) по содержательной линии «Алгоритмизация и программирование». Реализуйте эти программы на выбранном программном обеспечении.

Примерные темы:

а) уточнение понятия алгоритма;

б) свойства алгоритма;

в) исполнитель алгоритма, схема знакомств с исполнителем;

г) базовые управляющие команды организации действий в алгоритмах решения задач;

д) введение типов данных выбранного вами языка программирования;

е) выработка умений и навыков по составлению алгоритмов отобранных вами задач на данном этапе изучения информатики;

ж) использование игровых алгоритмов при введении основных понятий темы (игра «Баше», игра «Ханойская башня», игра «Жизнь» и др.);

з) имитация исполнения компьютером поиска элемента в неотсортированной таблице;

и) имитация исполнения компьютером поиска элемента в отсортированной таблице;

к) имитация исполнения компьютером простой сортировки элементов таблицы (выбор, обмен, простые включения).

Выполните задания 4-7. Проделанную и оформленную работу подготовьте для защиты. Проведите презентацию своих разработок в студенческой аудитории.

4. Составьте поурочный план изучения одной из выбранных вами тем содержательной линии «Алгоритмизация и программирование» на одной из ступеней непрерывного курса изучения информатики.

5. Разработайте содержание, методы и приёмы проведения лабораторных работ по одной из выбранных вами тем содержательной линии «Алгоритмизация и программирование» на каждой ступени непрерывного курса изучения информатики.

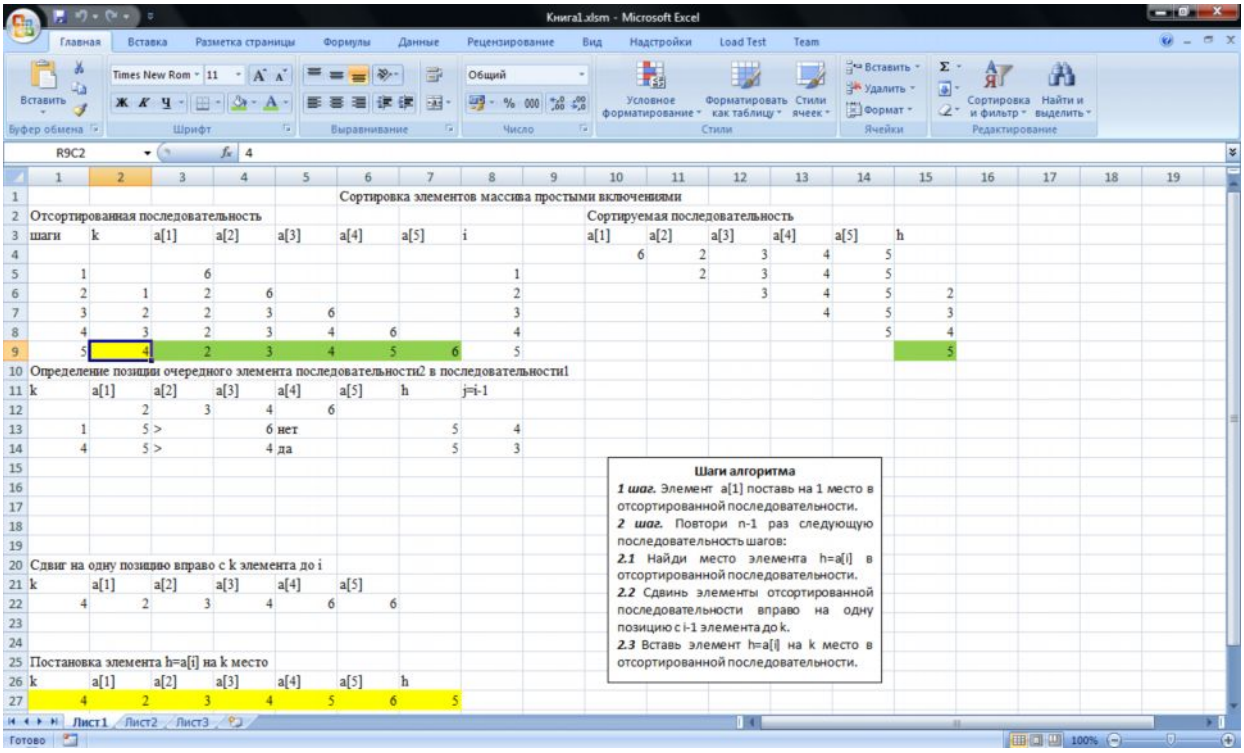
6. Разработайте содержание эвристических бесед на темы:

- *Криптография.* Когда и зачем нужно защищать информацию. Криптография – как наука о методах преобразования (шифрования) информации с целью её защиты от незаконных пользователей. Шифр – метод преобразования информации с целью её защиты от незаконных пользователей. Примеры классических шифров: шифр Цезаря, шифр «Сциталь», шифр Виженера, другие шифры.
- *Двоичное кодирование информации.* Сжатие информации. Классические методы сжатия информации при помощи кодов: код Фано, код Хаффмена и др. Современные методы сжатия информации: LZ77 и др.

7. Выполните задания для самостоятельной работы к параграфам главы 3.

Приложения

Приложение 1. Программа на языке VBA для Microsoft Excel, имитирующая механизм сортировки элементов массива простыми включениями




```
Sub Включение()
ThisWorkbook.Sheets(1).UsedRange.Delete
Cells(1, 6) = "Сортировка элементов массива простыми включениями"
Cells(2, 1) = "Отсортированная последовательность"
Cells(2, 10) = "Сортируемая последовательность"
Cells(3, 1) = "Шаги"
Cells(3, 2) = "k"
Cells(3, 8) = "i"
Cells(3, 15) = "h"
Cells(10, 1) = "Определение позиции очередного элемента последовательности2 в последовательности1"
Cells(11, 1) = "k"
```

```
Sub Место(i)
Dim z As Integer
Range("A13:H17").Delete (1): z = 13
Cells(z, 1) = 1: Cells(z, 8) = i - 1
For q = 1 To i
 Cells(12, q + 1) = Cells(i + 3, q + 2)
Next
While Cells(z, 1)=1 And Cells(z, 8)>=1
 Cells(i+3, 2).Interior.Color = xlNone
 MsgBox ("Ищем место элемента h=a[" & i & "]=" & Cells(i + 4, 15))
 Rows("10:30").Interior.Pattern=xlNone
 Cells(z, 7) = Cells(i + 4, 15)
 Cells(z, 7).Interior.Color = 65535
 Cells(z, 2) = Cells(z, 7)
 Cells(z, 2).Interior.Color = 65535
```

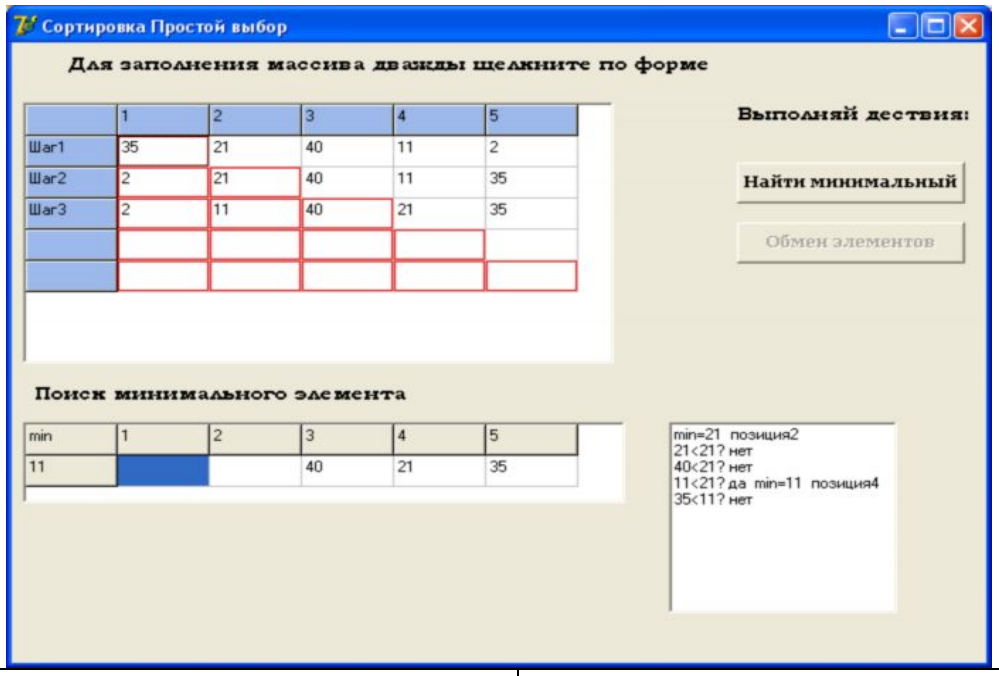
```
Cells(11, 7) = "h"
Cells(11, 8) = "j=i-1"
Cells(20, 1) = "Сдвиг на одну позицию вправо с k элемента до i"
Cells(21, 1) = "k"
Cells(25, 1) = "Постановка элемента h=a[i] на k место"
Cells(26, 1) = "k"
Cells(26, 7) = "h"
For i = 1 To 5
   Cells(3, 2 + i) = "a[" & i & "]"
   Cells(3, 9 + i) = "a[" & i & "]"
   Cells(11, i + 1) = "a[" & i & "]"
   Cells(21, i + 1) = "a[" & i & "]"
   Cells(26, i + 1) = "a[" & i & "]"
Next
For i = 10 To 14
   Cells(4, i) = InputBox("Введите a[" & i - 9 & "]", , , 7000, 7000)
Next
i = 1
Cells(4 + i, 1) = i
Cells(4 + i, 8) = i
Cells(4 + i, 3) = Cells(3 + i, 10)
Cells(i + 4, 3).Interior.Color = 5296274
For q = 1 To 5
   Cells(12, q + 1) = Cells(4, q + 1)
   Cells(22, q + 1) = Cells(4, q + 1)
   Cells(27, q + 1) = Cells(4, q + 1)
Next
For i = 2 To 5
   MsgBox ("Рассматриваем следующий элемент")
   Rows(2 + i).Interior.Pattern = xlNone
   For q = i To 5
      Cells(i + 3, q + 9) = Cells(i + 2, q + 9)
      Cells(i + 3, q + 9).Interior.Color =
 Cells(z, 3) = ">"
 Cells(z, 3).Interior.Color = 65535
 Cells(z, 4) = Cells(12, Cells(z, 8) + 1)
 Cells(z, 4).Interior.Color = 65535
 If Cells(z, 2) <= Cells(z, 4) Then
  Cells(z, 5) = "нет"
  Cells(z, 5).Interior.Color = 65535
  z = z + 1
  If Cells(z - 1, 8) - 1 > 0 Then
   Cells(z, 1) = Cells(z - 1, 1)
   Cells(z, 1).Interior.Color=65535
   Cells(z, 8) = Cells(z - 1, 8) - 1
   Cells(z, 8).Interior.Color=65535
                              Else
   MsgBox ("Нашли место элемента")
   Cells(i + 4, 2) = Cells(z - 1, 1)
   Cells(i + 4, 2).Interior.Color = 65535
  End If
                                  Else
  Cells(z, 5) = "да"
  Cells(z, 5).Interior.Color = 65535
  Cells(z, 1) = Cells(z, 8) + 1
  Cells(z, 1).Interior.Color = 65535
  MsgBox ("Нашли место элемента")
  Cells(i + 4, 2) = Cells(z, 1)
  Cells(i + 4, 2).Interior.Color = 65535
 End If
Wend
End Sub
Sub Сдвиг(i)
Rows("10:30").Interior.Pattern=xlNone
Cells(22, 1) = Cells(i + 4, 2)
For q = 1 To i
   Cells(22, q + 1) = Cells(i + 3, q + 2)
   Cells(22,q+1).Interior.Color = 65535
Next
MsgBox ("Сдвигаем элементы")
```

```
5296274
      Cells(i + 3, 15).Interior.Color = xlNone
   Next q
   Cells(i + 4, 15) = Cells(i + 3, i + 9)
   Cells(i + 4, 15).Interior.Color = 5296274
   Cells(4 + i, 1) = i
   Cells(4 + i, 8) = i
   Место (i)
   Сдвиг (i)
   Постановка (i)
   For q = 1 To i
      Cells(i + 4, q + 2) = Cells(27, q + 1)
      Cells(i + 4, q + 2).Interior.Color = 5296274
   Next
   Rows(2 + i).Interior.Pattern = xlNone
Next
Rows(2 + i).Interior.Pattern = xlNone
MsgBox ("СОРТИРОВКА ОКОНЧЕНА")
End Sub
```

```
For q = i To Cells(22, 1) + 1 Step -1
   Cells(22, q + 1) = Cells(22, q)
   Cells(22,q+1).Interior.Color = 65535
Next
MsgBox ("Сдвинули элементы")
End Sub
Sub Постановка(i)
Rows("10:30").Interior.Pattern=xlNone
Cells(27, 1) = Cells(22, 1)
Cells(27, 7) = Cells(4 + i, 15)
Cells(27, 1).Interior.Color = 65535
Cells(27, 7).Interior.Color = 65535
For q = 1 To i
   Cells(27, q + 1) = Cells(22, q + 1)
   Cells(27,q+1).Interior.Color = 65535
Next q
MsgBox ("Вставляем элемент")
Cells(27,Cells(27, 1) + 1) = Cells(27, 7)
Cells(27, Cells(27, 1) + 1).Interior.Color = 65535
MsgBox ("Вставили элемент")
End Sub
```

Процедура, демонстрирующая сущность сортировки элементов массива простыми включениями

Приложение 2. Процедура на языке Turbo Delphi, имитирующая механизм сортировки элементов массива простым выбором




```
var
  Form1: TForm1;
var n:integer;
    a:array[1..100] of integer;
    min,nmin,i,j:integer;
implementation
{$R *.dfm}
procedure TForm1.FormDblClick(Sender:
TObject);
var s:string;   var i:integer;
begin
Button1.Enabled:=False; But-
ton2.Enabled:=False;
s:=InputBox('Введите размер массива',
'','');
n:=StrToInt(s);
StringGrid1.ColCount:=n+1;
StringGrid2.ColCount:=n+1;
for i:=1 to n do
  begin
  StringGrid1.Cells[i,0]:=IntToStr(i);
  StringGrid2.Cells[i,0]:=IntToStr(i);
  end;
```

```
for j:=i to n do
    begin
RichE-
dit1.Text:=RichEdit1.Text+IntToStr(a[j])+'
<'+IntToStr(min)+'?';
    if a[j]<min then
      begin
      min:=a[j]; nmin:=j;
      RichEdit1.Text:=RichEdit1.Text+' да
min='+IntToStr(a[j])+'
позиция'+IntToStr(nmin)+Chr(13);
      StringGrid2.Cells[0,1]:=IntToStr(min);
      end
    else
      RichEdit1.Text:=RichEdit1.Text+'
нет'+Chr(13);
    end;
    otv:='Минимальный элемент
'+IntToStr(min)+ 'на позиции
'+IntToStr(nmin);
  ShowMessage(otv);
  end
else
```

```
StringGrid1.Cells[0,1]:='Шаг1';
for i:=1 to n do
  begin
  s:=InputBox('Введите элемент массива',
'','');
  a[i]:=StrToInt(s);
  StringGrid1.Cells[i,1]:=IntToStr(a[i]);
  end;
StringGrid2.Cells[0,0]:='min';
Button1.Enabled:=True;
end;
procedure TForm1.FormCreate(Sender:
TObject);
begin
i:=0;
end;
procedure TForm1.Button1Click(Sender:
TObject);
  var otv:string;
begin
Button2.Enabled:=True;
Button1.Enabled:=False;

if i<n-1 then
  begin
  i:=i+1;
  for j:=1 to n do
    StringGrid2.Cells[j,1]:='';
  for j:=i to n do
    StringGr-
id2.Cells[j,1]:=StringGrid1.Cells[j,i];
  min:=a[i]; nmin:=i;
  StringGrid2.Cells[0,1]:=IntToStr(min);
  RichEdit1.Text:='min='+IntToStr(a[i])+'
позиция'+IntToStr(nmin)+Chr(13);
  begin
  ShowMessage('Массив отсортирован');
  for j:=1 to n do
    StringGrid2.Cells[j,1]:='';
  Button1.Enabled:=false;
  Button2.Enabled:=False;
  end;
end;
procedure TForm1.Button2Click(Sender:
TObject);
var c:integer; var j:integer;  otv:string;
begin
Button1.Enabled:=True;
Button2.Enabled:=False;
otv:='Меняем местами элементы
'+IntToStr(a[i])+' и '+IntToStr(a[nmin]);
ShowMessage(otv);
c:=a[i]; a[i]:=a[nmin]; a[nmin]:=c;
StringGrid1.RowCount:=n+1;
StringGr-
id1.Cells[0,i+1]:='Шаг'+IntToStr(i+1);
for j:=1 to n do
  StringGrid1.Cells[j,i+1]:=IntToStr(a[j]);
for j:=1 to n do
    StringGrid2.Cells[j,1]:='';
  for j:=i+1 to n do
    StringGrid2.Cells[j,1]:=IntToStr(a[j]);
end;
procedure
TForm1.StringGrid1DrawCell(Sender:
TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
  var q,w:Integer;
begin
StringGrid1.Canvas.Brush.Color:=clRed;
for q:=1 to n do
  for w:=1 to q do
    if (ACol=w) and (ARow=q) then
      StringGrid1.Canvas.FrameRect(Rect);
end;
end.
```

Процедура, демонстрирующая сущность сортировки элементов массива простым выбором

Библиографический список

1. Авербух, А.В. Изучение основ информатики и вычислительной техники: Пособие для учителя / А.В. Авербух, В.Б. Гисин, Я.Н. Зайдельман, Г.В. Лебедев. – М.: Просвещение, 1992. – 302 с. – ISBN 5-09-002845-1.

2. Андреева, Е.В. Математические основы информатики. Элективный курс: методическое пособие / Е.В. Андреева, Л.Л. Босова, И.Н. Фалина – М.: БИНОМ. Лаборатория знаний, 2007. – 312 с. – ISBN 5-94774-138-5.

3. Босова, Л.Л. Информатика и ИКТ. 5-7 классы: методическое пособие / Л.Л. Босова, А.Ю. Босова. – М.: БИНОМ. Лаборатория знаний, 2011. – 479 с. – ISBN 978-5-9963-0457-8.

4. Босова, Л.Л. Информатика: учебник для 6 класса / Л.Л. Босова. – М.: БИНОМ. Лаборатория знаний, 2008. – 208 с. – ISBN 978-5-94774-836-9.

5. Босова, Л.Л. Информатика: учебник для 7 класса / Л.Л. Босова. – М.: БИНОМ. Лаборатория знаний, 2008. – 229 с. – ISBN 978-5-94774-834-5.

6. Гарднер, М. Крестики-нолики / М. Гарднер. – М.: Мир, 1988. – 352 с. – ISBN 5-03-001234-6.

7. Информатика и ИКТ. Задачник-практикум: в 2 т. Т. 1 / Л.А. Залогова [и др.]; под ред. И.Г. Семакина, Е.К. Хеннера. – М.:БИНОМ. Лаборатория знаний, 2011. – 309 с. ISBN 978-5-9963-0476-9 (Т. 1), ISBN 978-5-9963-0475-2.

8. Информатика и ИКТ. Задачник-практикум: в 2 т. Т. 2 / Л.А. Залогова [и др.]; под ред. И.Г. Семакина, Е.К. Хеннера. – М.:БИНОМ. Лаборатория знаний, 2011. – 294 с. – ISBN 978-5-9963-0477-9 (Т. 2), ISBN 978-5-9963-0475-2.

9. Информатика и ИКТ. Практикум 8-9 класс / под ред. проф. Н.В Макаровой. – СПб.: Питер, 2008. – 384 с. – ISBN 978-5-469-01622-9.

10. Информатика и ИКТ. Учебник. Начальный уровень / под ред. проф. Н.В. Макаровой. – СПб.: Лидер, 2010. – 160 с. – ISBN 978-5-91180-197-7.

11. Информатика. 7-9 класс. Базовый курс. Теория / под ред. проф. Н.В. Макаровой. – СПб.: Питер, 2008. – 384 с. ISBN 5-272-00186-9.

12. Информатика. Методическое пособие для учителей. 7 класс / под ред. проф. Н.В. Макаровой. – СПб.: Питер, 2003. – 384 с. – ISBN 5-94723-636-2.

13. Кушниренко, А.Г. Информатика. 7-9 кл. :учеб. для общеобразоват. учеб. заведений / А.Г. Кушниренко, Г.В. Лебедев, Я.Н. Зайдельман. – М.: Дрофа, 2000. – 336 с. – ISBN 5-7107-3109-9.

14. Лапчик, М.П. Методика преподавания информатики: учеб. пособие для студ. пед. Вузов / М.П. Лапчик, И.Г. Семакин, Е.К. Хеннер; под общей ред. М.П. Лапчика. – М.: Издательский центр «Академия», 2001. – 624 с. – ISBN 5-7695-0825-6.

15. Лапчик, М.П. Теория и методика обучения информатике / М.П. Лапчик, М.И.Рагулина, Н.Н.Самылкина, И. Г.Семакин, Е.К. Хеннер. – М.: Издательство Academia, 2008. – 592 с. – ISBN: 978 -5-7695-4748-5.

16. Могилев, А.В. Информатика: учеб. пособие для студ. пед. вузов / А.В. Могилев, Н.И. Пак, Е.К. Хеннер; под ред. Е.К. Хеннера. – М.: Издательский центр «Академия», 2004. – 848 с. – ISBN 5-7695-1709-3.

17. Николаева, И.В. Алгоритмизация и программирование. Ч. 1 / И.В. Николаева, Е.П. Давлетярова. – Владимир: ВГПУ, 2006. – 84 с.

18. Николаева, И.В. Алгоритмизация и программирование. Ч. 3 / И.В. Николаева, Е.П. Давлетярова. – Владимир: ВГГУ, 2010. – 48 с.

19. Николаева, И.В. Алгоритмизация и программирование. Ч. 4 / И.В. Николаева, Е.П. Давлетярова. – Владимир: ВГГУ, 2010. – 52 с.

20. Николаева, И.В. Алгоритмизация и программирование. Ч. 6 / И.В. Николаева, Е.П. Давлетярова. – Владимир: ВГГУ, 2010. – 48 с.

21. Николаева, И.В. Формализация и моделирование. Ч. 1 / И.В. Николаева. – Владимир: ВГПУ, 2003. – 80 с.

22. Николаева, И.В. Численные методы и компьютерное моделирование / И.В. Николаева. – Владимир: ВГПУ, 2005. – 62 с.

23. Семакин, И.Г. Информатика и ИКТ. Базовый уровень: практикум для 10-11 класса / И.Г. Семакин, Е.К. Хеннер, Т.Ю. Шеина. – М.: БИНОМ. Лаборатория знаний, 2011. – 120 с. – ISBN 978-5-9963-0596-4.

24. Семакин, И.Г. Информатика и ИКТ. Профильный уровень: учебник для 10 класса / И.Г. Семакин, Т.Ю. Шеина, Л.В. Шестакова. – М.: БИНОМ. Лаборатория знаний, 2010. – 363 с. – ISBN 978-5-9963-0325-0.

25. Семакин, И.Г. Информатика. Базовый курс. 7-9 классы / И.Г. Семакин, Л.А. Залогова, С.В. Русаков, Л.В. Шестакова – М.: Лаборатория базовых знаний, 2003. – 390 с. – ISBN 5-94774-082-6.

26. Угринович, Н.Д. Информатика и ИКТ. Профильный уровень: учебник для 11 класса / Н.Д. Угринович. – М.: БИНОМ. Лаборатория знаний, 2010. – 308с. – ISBN 978-5-99663-0328-1.

27. Угринович, Н.Д. Информатика и ИКТ. Профильный уровень: учебник для 10 класса / Н.Д. Угринович. – М.: БИНОМ. Лаборатория знаний, 2008. – 387с. – ISBN 978-5-94774-828-4.

28. Фаронов, В.В. Турбо Паскаль 7.0. Начальный курс: учебное пособие / В.В. Фаронов. – М.: КНОРУС, 2006. – 576 с. – ISBN 5-85971-138-7.

29. Федоренко, Ю. Алгоритмы и программы на QBasic. Учебный курс / Ю. Федоренко. – СПб.: Питер, 2002. – 288 с. – ISBN 5-318-00693-0.

30. Юдина, А.Г. Практикум по информатике в среде LogoWriter: Пособие для учащихся общеобразовательных учреждений / А.Г. Юдина. – М.: Мнемозина, 1999. – 127 с. – ISBN 5-87441-142-9.

Учебное издание

НИКОЛАЕВА Ирина Васильевна
ДАВЛЕТЯРОВА Елена Петровна

ТЕОРИЯ И МЕТОДИКА ОБУЧЕНИЯ ИНФОРМАТИКЕ. СОДЕРЖАТЕЛЬНАЯ ЛИНИЯ «АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ»

Учебное пособие

Компьютерный набор: Е.П. Давлетярова,
И.В. Николаева

Подписано в печать 30.05.12.
Формат 60x84/16. Усл. печ. л. 13,25. Тираж 500 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87