

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Государственное образовательное учреждение высшего
профессионального образования**

Северо-Западный государственный заочный технический университет

М. П. Белов

Основы алгоритмизации в информационных системах

Утверждено редакционно-издательским советом университета
в качестве учебного пособия

Санкт-Петербург

2003

Белов М. П. Основы алгоритмизации в информационных системах: Учеб. пособие. - СПб.: СЗТУ, 2003. - 85 с.

Пособие соответствует требованиям государственных образовательных стандартов высшего профессионального образования по направлению подготовки дипломированного специалиста 651900 (специальность 210100 - "Управление и информатика в технических системах") и направлению подготовки бакалавра 550200.

В пособии рассмотрены понятие алгоритма, способы описания алгоритмов, основные свойства и характеристики алгоритмов, алгоритмические структуры и правила их оформления, методы разработки алгоритма, приводятся примеры.

Пособие предназначено для студентов факультета информатики и систем управления, но может быть полезно и для студентов других специальностей.

Рецензенты: В. И. Репкин, канд. техн. наук, доцент кафедры "Электротехники, вычислительной техники и автоматизации" Петербургского института машиностроения, А. Ю. Дорогов канд. техн. наук, доцент кафедры АПУ факультета компьютерных технологий и информатики Санкт-Петербургского государственного электротехнического университета.

Введение

Любой человек ежедневно встречается с множеством задач от самых простых и хорошо известных до очень сложных. Для многих задач существуют определенные правила (инструкции, предписания), объясняющие исполнителю, как решать данную задачу. Эти правила человек может изучить заранее или сформулировать сам в процессе решения задачи. Чем точнее и понятнее будут описаны правила решения задач, тем быстрее человек овладеет ими и будет эффективнее их применять. Источниками возникновения алгоритмов служат: наблюдение и эксперимент, научная теория, прошлый опыт и др.

Решение многих задач человек может передавать техническим устройствам – ПК, автоматам, роботам и т.д. Применение технических устройств предъявляет очень строгие требования к точности описания правил и последовательности выполнения действий. Поэтому разрабатываются специальные языки для четкого и строгого описания различных правил.

Алгоритмизация это раздел информатики, изучающий методы и приемы построений алгоритма, а также их свойства, т. е. алгоритмика задачи, построения модели и алгоритмизация.

По содержанию пособие условно делится на две главы. Первая глава имеет общетеоретическую направленность. В ней излагаются понятие алгоритма, способы описания алгоритмов, основные свойства и характеристики алгоритмов, алгоритмические структуры и правила их оформления, методы разработки алгоритмов. Во второй главе приводятся примеры алгоритмов и программ, написанных на алгоритмическом языке Borland C++ решения различных задач.

Сложность изучения этого раздела информатики состоит в том, что она охватывает очень широкий круг вопросов и для поиска ответов на них приходится просматривать слишком много учебной и специальной литературы. В разных источниках уровень изложения различен. В данном пособии весь материал адаптирован к уровню подготовки студентов. Благодаря этому каждый студент имеет возможность полностью разобраться в прочитанном материале, не прибегая к другим пособиям и посторонней помощи.

Для определения степени усвоения материала, в конце каждой главы приводится список вопросов для самопроверки и упражнения.

При написании учебного пособия автор сосредоточил усилие на подборе необходимого материала и изложении его в доступной форме. Большинство

теоретических положений в пособии (но не конкретных примеров и расчетов) заимствованы из различных опубликованных источников (см. список литературы).

В заключение автор выражает признательность рецензентам и редактору за участие в подготовке данного учебного пособия, которое, вероятно, будет полезно не только студентам специальности 210100, но и обучающимся по другим специальностям.

Глава 1

АЛГОРИТМИЧЕСКИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ

1.1. Понятие алгоритма

Для решения задачи исполнителю необходимо указать последовательность действий, которые он должен выполнить для достижения цели – получения результата. Иначе говоря, исполнителю должен быть указан алгоритм решения задачи, представленный на понятном ему языке. Под исполнителем подразумевается как человек, так и вычислительная машина.

Перед решением любой задачи с помощью персонального компьютера (ПК) выполняются следующие этапы: постановка этой задачи, построение сценария и алгоритмизация.

Алгоритмизация задачи – процесс разработки (проектирования) алгоритма решения задачи с помощью ПК на основе ее условия и требований к конечному результату.

На этапе постановки задачи описываются исходные данные и предпосылки, формируются правила начала и окончания решения задачи (достижения цели), т. е. разрабатывается информационная или эквивалентная ей математическая модель. Методом проб и ошибок ведется поиск метода решения задачи (метода вычислений, метода перебора вариантов, метода распознавания образов). На основании этого метода разрабатывается исходный алгоритм, реализация которого принципиально возможна с помощью ПК. При разработке исходного алгоритма и даже при выборе модели пользователь, т. е. человек, решающий конкретную задачу, должен иметь представление о математическом обеспечении ПК.

Алгоритм – понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи [1].

Алгоритм применительно к ПК – точное предписание, т.е. набор операций и правил их чередования, при помощи которого, начиная с некоторых исходных данных, можно решить задачу фиксированного типа. Команда алгоритма – предписание о выполнении отдельного законченного действия исполнителя.

Термин алгоритм происходит от имени узбекского ученого IX в. Аль-Хорезми, который в своем труде "Арифметический трактат", переведенном в XII в. с арабского на латынь, изложил правила арифметических действий над

числами в позиционной десятичной системе счисления. Эти правила и называли алгоритмами. Таким образом, правила сложения, вычитания, деления, умножения чисел, правила преобразования алгебраических выражений, правила построения геометрических фигур, грамматические правила правописания слов и предложений – все это алгоритмы. Многие правила, инструкции, записанные в различных документах и представляющие собой подробнейшие указания, годные во всевозможных ситуациях, также можно отнести к алгоритмам.

Виды алгоритмов как логико-математических средств отражают также компоненты человеческой деятельности, а сами алгоритмы в зависимости от цели, начальных условий задачи, путей ее решения и определения действий исполнителя подразделяются на [3]:

- механические алгоритмы, или детерминированные, жесткие (например, алгоритм работы машины, двигателя и т. п.);
- гибкие алгоритмы, например стохастические, т. е. вероятностные и эвристические.

Механический алгоритм задает определенные действия, обозначая их в единственной и достоверной последовательности, обеспечивая тем самым однозначный требуемый или искомый результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм.

Вероятностный (стохастический) алгоритм дает программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата.

Эвристический алгоритм (от греческого слова "эврика" – "Я нашел") – это такой алгоритм, в котором достижение конечного результата программы действий однозначно не предопределено, так же как не обозначена вся последовательность действий, не выявлены все действия исполнителя. К эвристическим алгоритмам относят, например, инструкции и предписания.

В этих алгоритмах используются универсальные логические процедуры и способы принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения схожих задач.

Эвристика (в переводе с греческого – отыскивают, открывают) – это совокупность специальных методов и приемов, позволяющих открыть новое, неизвестное, найти решение нетривиальной задачи.

Эвристика изучает продуктивное творческое мышление и на этой основе выявляет способы построения оптимальных направлений поиска решений задач,

точные методы решения которых неизвестны.

Часто, для получения новых алгоритмов, используются уже существующие алгоритмы. Это осуществляется комбинированием уже известных алгоритмов или с помощью эквивалентных преобразований алгоритмов.

Алгоритмы называются эквивалентными, если результаты, получаемые с помощью этих алгоритмов для одних и тех же исходных данных, одинаковы.

Типичный пример эквивалентного преобразования алгоритмов – перевод с одного алгоритмического языка на другой.

В общем случае алгоритмизация вычислительного процесса включает следующие действия:

- 1) последовательную декомпозицию задачи, выделение автономных этапов вычислительного процесса и разбивку каждого этапа на отдельные шаги;
- 2) формальную запись содержания каждого этапа и/или шага;
- 3) определение общего порядка выполнения этапов и/или шагов;
- 4) проверку правильности алгоритма.

Последовательная декомпозиция предполагает разделение сложной задачи на совокупность более простых подзадач.

Часто начинающие программисты не уделяют этапу алгоритмизации достаточного внимания и даже пытаются его игнорировать. В результате процесс программирования сильно усложняется.

Значительно проще решать задачу постепенно, в два этапа (при этом сложность выполнения каждого отдельного этапа получается в несколько раз меньше сложности исходной задачи).

На первом этапе надо наметить общую стратегию решения задачи и составить соответствующий алгоритм. Причем для сложной задачи алгоритмизация выполняется постепенно. Сначала составляется укрупненная схема решения, а затем схемы работы отдельных блоков. Кроме того, при алгоритмизации одного и того же процесса можно использовать несколько способов записи (начиная с менее формализованных форм).

На втором этапе остается лишь выполнить кодирование (программирование), заменив формульно-словесные инструкции алгоритма операторами конкретного языка. Эта работа уже не связана с большим умственным напряжением. При несложных задачах для ее выполнения достаточно знать общие правила оформления программ, правила описания данных, основные операторы (ввода/вывода, обработки, управления).

Если бы мы знали алгоритмы решения всех задач, то их исполнение можно было бы поручить машине. Но оказалось, что не все задачи, которые нам хотелось бы решить, имеют алгоритмы решения. Задачи, в принципе не имеющие общего решения, называют алгоритмически неразрешимыми. К примеру, мы знаем, как решить любое квадратное алгебраическое уравнение, пользуясь одним и тем же алгоритмом. Похожие формулы существуют и для кубических уравнений и для уравнений четвертой степени. Но уже для уравнений степени выше четвертой таких формул нет и в принципе быть не может, хотя в частном случае отдельные уравнения можно решить. Есть и другие алгоритмически неразрешимые задачи, например задача о трисекции угла, о квадратуре круга и др.

1.2. Свойства алгоритмов

Алгоритмы обладают целым рядом свойств [1]: понятностью, дискретностью, точностью, результативностью, массовостью.

Свойства алгоритма – набор свойств, отличающих алгоритм от любых предписаний и обеспечивающих его автоматическое исполнение.

Понятность для исполнителя – содержание предписания о выполнении только таких действий, которые входят в систему команд исполнителя, т. е. алгоритм должен быть задан с помощью таких указаний, которые исполнитель (персональный компьютер, промышленный компьютер, контроллер, однокристалльная микроЭВМ и др.) может воспринимать и выполнять по ним требуемые действия (операции).

Дискретность (прерывность, раздельность) – выполнение команд алгоритма последовательно, с точной фиксацией моментов окончания выполнения одной команды и начала выполнения следующей, т. е. алгоритм должен содержать последовательность указаний (команд), каждое из которых приводит к выполнению в исполнителе одного шага (действия).

Определенность – каждое правило алгоритма должно быть четким, однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

Результативность – либо завершение решения задачи после выполнения алгоритма, либо вывод о невозможности продолжения решения по какой-либо из причин, т. е. алгоритм должен обеспечивать возможность получения резуль-

тата после конечного числа шагов.

Массовость – означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

1.3. Основные характеристики алгоритмов

Для решения одной и той же задачи как правило можно использовать различные алгоритмы. В связи с этим, возникает необходимость сравнивать их между собой, и для этого нужны определенные критерии качества алгоритмов.

Временные характеристики алгоритма определяют длительность решения или временную сложность [4].

Длительность решения часто выражается в единицах времени, но удобнее ее выражать через количество операций, так как количество операций не зависит от быстродействия конкретной машины.

Временной сложностью алгоритма называется зависимость времени счета, затрачиваемого на получение результатов от объема исходных данных.

Временная сложность позволяет определить наибольший размер задачи, которую можно решить с помощью данного алгоритма на ПК. Каждый алгоритм можно характеризовать функцией $f(n)$, выражающей скорость роста объема вычислений при увеличении размерности задачи – n . Если эта зависимость имеет линейный или полиномиальный характер, то алгоритм считается "хорошим", если экспоненциальный – "плохим".

Для сложных задач эта характеристика имеет большое значение, т.к. ее изменение значительно сильнее влияет на время решения, чем изменение быстродействия ПК. Например, при зависимости $f(n) = 2^n$ увеличение производительности в 10 раз увеличивает размерность задачи, решаемой за то же время, всего на 15% [4].

Объемные характеристики алгоритма определяют его информационную сложность. Информационная сложность связана со сложностью описания, накопления и хранения исходных, промежуточных и результирующих данных при решении определенной задачи.

Объем текста алгоритма (программы) определяется количеством операто-

ров, использованных для записи алгоритма.

Объем внутренней и внешней памяти необходимой для хранения данных и программ при использовании данного алгоритма определяется на основании расчетов или опытным путем. При недостатке памяти носителей информации используется сегментация программ.

Сложность структуры алгоритма определяется количеством маршрутов, по которым может реализовываться процесс вычислений и сложностью каждого маршрута.

Очевидно, что при выборе алгоритмов нужно учитывать не только их характеристики качества, но и способ реализации алгоритма. Например, многие итерационные алгоритмы удобны для ПК, но слишком трудоемки для человека. Тип используемой ПК также может влиять на выбор алгоритма (иногда имеет место и обратный вариант, когда сначала определяется алгоритм и лишь затем способ реализации).

1.4. Способы описания алгоритмов

Для строгого задания различных структур данных и алгоритмов их обработки требуется иметь такую систему формальных обозначений и правил, чтобы смысл всякого используемого предписания трактовался точно и однозначно. Соответствующие системы правил называют языками описаний.

К средствам описания алгоритмов относятся следующие основные способы их представления: словесный; графический; псевдокоды; программный. На практике используются также и другой способ описания: табличный (таблицы переключений (таблицы истинности); таблицы автоматов; циклограммы работы; таблицы решений).

1.4.1. Словесный способ представления алгоритмов

Словесный способ записи алгоритмов представляет собой последовательное описание основных этапов обработки данных и задается в произвольном изложении на естественном языке.

В качестве примера рассмотрим запись алгоритма нахождения наибольшего общего делителя двух натуральных чисел (m и n). Алгоритм может быть записан в следующем виде:

- если числа равны, то необходимо взять любое из них в качестве ответа, в противном случае – продолжить выполнение алгоритма;

- определить большее из чисел;
- заменить большее число разностью большего и меньшего чисел;
- повторить алгоритм сначала.

Способ основан на использовании общепринятых средств общения между людьми и с точки зрения написания трудностей не представляет. Такой способ записи удобно использовать на начальном этапе алгоритмизации задачи. К недостаткам словесного способа записи можно отнести следующее: 1) полное подробное словесное описание алгоритма получается очень громоздким; 2) естественный язык допускает неоднозначность толкования отдельных инструкций; 3) при переходе к этапу программирования требуется дополнительная работа по формализации алгоритма, так как словесное описание может быть понятно человеку, но "непонятно" ПК. Поэтому словесный способ записи алгоритмов не имеет широкого распространения.

1.4.2. Графический способ представления алгоритмов

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

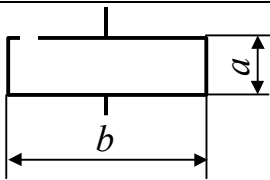
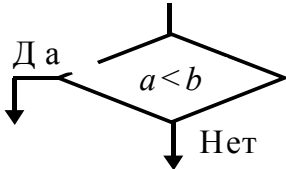
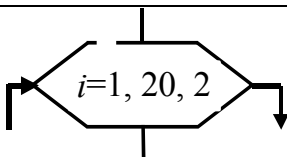
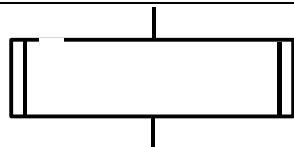
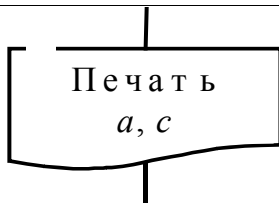
Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т. п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. Для начертания этих схем используется набор символов, определяемых ГОСТ 19.701–90 (ИСО 5807 – 85) [2] "Единая система программной документации". В табл. 1 приведены наиболее часто употребляемые символы.

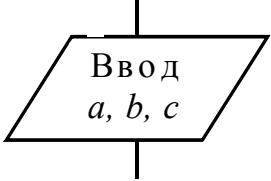

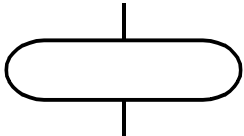

Символ "Процесс" применяется для обозначения одного или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединить в один блок. Представление отдельных операций достаточно свободно. Например, для обозначения вычислений можно использовать математические выражения, для пересылок данных – стрелки, для

других действий – пояснения на естественном языке. В зависимости от уровня детализации схемы пояснения на естественном языке могут быть более или менее подробными. Метод блок-схем, так же как и алгоритмический язык (псевдокод), независим от специфики языков программирования, поэтому в описаниях операторов не следует использовать резервированные слова и символы языков программирования, а также применять имена данных, образованные в соответствии с синтаксическими правилами этих языков.

Символ "Решение" используется для обозначения переходов управления по условию. В каждом блоке решения должны быть указаны вопрос, решение, условие или сравнение, которые он определяет.

Таблица 1

Название символа	Обозначение	Пояснение
1	2	3
Процесс		Вычислительное действие или последовательность вычислительных действий
Решение		Проверка условий
Модификация		Начало цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Документ		Вывод, печать результатов на бумаге

1	2	3
Ввод - вывод		Ввод – вывод данных в общем виде
Соединитель		Разрыв линий потока
Пуск, останов		Начало, конец, останов, вход и выход в подпрограммах
Комментарий		Пояснения, содержание подпрограмм, формулы

Стрелки, выходящие из блока решения, должны быть помечены соответствующими ответами (например, ДА, НЕТ), так чтобы были учтены все возможные ответы.

Символ "Модификация" используется для выполнения операций, меняющих команды или группы команд, изменяющих программу (например, для организации циклических конструкций). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и правило изменения значения параметра для каждого повторения. Блок размещается в начале циклической конструкции, для управления которой он используется, даже в том случае, если изменение параметра и проверка условий окончания цикла при реализации алгоритма производится не в начале, а в конце цикла.

Линии переходов используются для обозначения порядка выполнения действий. Для улучшения наглядности следует придерживаться стандартных правил изображения линий передач управления – сверху вниз и слева направо. Если необходимо показать передачу управления снизу вверх или справа налево, то направление следует отметить стрелкой.

Символ "Предопределенный процесс" используется для указания обращений к вспомогательным алгоритмам, выделенным автономно, в виде некоторого модуля; для обращений к библиотечным подпрограммам; для обозначения части алгоритма, не зависящей от основной схемы управления; для обозначения определенной части алгоритма, которая будет кодироваться вместе со

всем алгоритмом, но в документации представлена отдельной схемой. Если такая часть алгоритма представляет собой итерационный процесс, то в соответствующий ей блок вызова необходимо включить описания условий окончания цикла. По мнению некоторых специалистов, использование более одной схемы для одного алгоритма затрудняет его понимание. Однако практика показывает, что удобнее всего применять схемы алгоритмов, разбитые в соответствии с уровнями абстракции.

Символ "Документ" предназначен для ввода – вывода данных, носителем которых служит бумага.

Символ "Ввод - вывод" используется для преобразования данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Отдельным логическим устройствам ПК или отдельным функциям обмена соответствуют определенные блочные символы. В каждом из них указываются тип устройства или файла данных, тип информации, участвующий в обмене, а также вид операции обмена.

Символ "Соединитель" используется в том случае, когда схема алгоритма разделяется на автономные части, особенно если она не уместится на одном листе, или когда необходимо избежать излишних пересечений линий переходов. Применение соединителей не должно нарушать структурности при изображении схем.

Символ "Пуск - останов" используется для обозначения начала, конца, прерывания процесса обработки данных или выполнения программы.

Символ "Комментарий" позволяет включать в схемы алгоритмов пояснения к функциональным блокам. Частое использование комментариев нежелательно, так как это усложняет (загромождает) схему, делает ее менее наглядной. Однако некоторые обозначения переменных, принятые допущения или назначение отдельных алгоритмов требуют пояснительных записей.

1.4.2.1. Правила выполнения схем

Для облегчения вычерчивания и нахождения на схеме символов рекомендуется поле листа разбивать на зоны. Размеры зон устанавливают с учетом минимальных размеров символов, изображенных на данном листе. Допускается один символ размещать в двух и более зонах, если размер символа превышает размер зоны.

Координаты зоны проставляют: по горизонтали – арабскими цифрами

слева направо в верхней части листа; по вертикали – прописными буквами латинского алфавита сверху вниз в левой части листа.

Координаты зон в виде сочетания букв и цифр присваивают символам, вписанным в поля этих зон, например A1, A2, A3, B1, B2, B3 и т. д.

При выполнении схем от руки, если поле листа не разбито на зоны, символам присваивают порядковые номера.

В пределах одной схемы, при выполнении ее от руки, допускается применять не более двух смежных размеров ряда чисел, кратных 5.

Для ускорения выполнения схем от руки рекомендуется использовать бланки с контуром прямоугольника внутри каждой зоны. Контурные не должны воспроизводиться при изготовлении копии.

Расположение символов на схеме должно соответствовать требованиям ГОСТ [2].

Исключение составляют обязательные символы "Линия потока", "Канал связи", "Комментарий" и рекомендуемые символы "Межстраничный соединитель", "Транспортирование носителей", "Материальный поток".

Линии потока должны быть параллельны линиям внешней рамки схемы. Направления линии потока сверху вниз и слева направо принимают за основные и, если линии потока не имеют изломов, стрелками можно не обозначать. В остальных случаях направление линии потока обозначать стрелкой обязательно.

Расстояние между параллельными линиями потока должно быть не менее 3 мм, между остальными символами схемы – не менее 5 мм.

Записи внутри символа или рядом с ним должны выполняться машинописью с одним интервалом или чертежным шрифтом.

Записи внутри символа или рядом с ним должны быть краткими. Сокращения слов и аббревиатуры, за исключением установленных государственными стандартами, должны быть расшифрованы в нижней части поля схемы или в документе, к которому эта схема относится.

Для удобства детализации программы должны быть использованы символы "Процесс", "Решение", "Модификация", "Ввод – вывод" и "Пуск – останов".

Записи внутри символа должны быть представлены так, чтобы их можно было читать слева направо и сверху вниз, независимо от направления потока.

В схеме символу может быть присвоен идентификатор, который должен помещаться слева над символом (например, для ссылки в других частях документации)

В схемах допускается краткая информация о символе (описание, уточнение или другие перекрестные ссылки для более полного понимания функции данной части системы). Описание символа должно помещаться справа над символом

1.4.2.2. Соотношение геометрических элементов символов

Размер a должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер a на число, кратное 5. Размер b равен $1,5 \cdot a$. При ручном выполнении схем алгоритмов и программ для символов, представленных в табл. 1, допускается устанавливать b равным $2 \cdot a$.

При выполнении условных графических обозначений автоматизированным методом размеры геометрических элементов символов округляются до значений, определяемых техническими возможностями используемых устройств.

1.4.2.3. Графические способы описания алгоритмов работы информационных систем (промышленных систем)

К графическим способам описания алгоритмов работы информационных систем (промышленных систем) относятся также:

- Диаграммы. Применяются для описания зависимости входных и выходных переменных состояния друг от друга или от времени.

- Диаграммы последовательности включений и пошаговые диаграммы перемещений. Применяются для описания линейных (неразветвленных) дискретных процессов, например работы простых станков. На оси ординат указываются команды включения и состояния системы. На оси абсцисс – такты работы или реальный масштаб времени.

- Диаграммы работы. Применяются для описания работы контакторных схем управления.

- Схемы блокировки. Схемы блокировки (рис. 1)

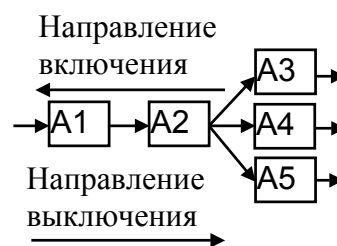


Рис. 1

служат для изображения предусмотренной технологией взаимосвязанности процессов включения и выключения.

- Структурные схемы (рис. 2, а) и сигнальные графы (рис. 2, б). Применяются для изображения структуры и описания функционирования преимущественно непрерывных систем. Переход от одной формы описания к другой очень прост. Передаточные функции (передачи) F_i отдельных функциональных блоков записываются в структурной схеме внутри соответствующих прямоугольников, а в направленном графе – на его ветвях (ребрах). Переменным (сигналам) x_i в структурной схеме соответствуют линии, соединяющие блоки, а в графе – его узлы (вершины).

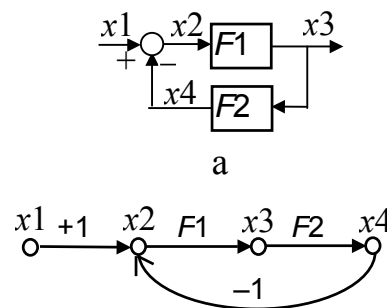


Рис. 2

- Автоматные графы (рис. 3). Применяются для описания дискретных состояний системы и возможных переходов между ними. Узлы изображают различные возможные состояния q_i , а ветви со стрелками – переходы. Рядом с каждой ветвью записывается условие B_j , которое вызывает переход между соответствующими состояниями.

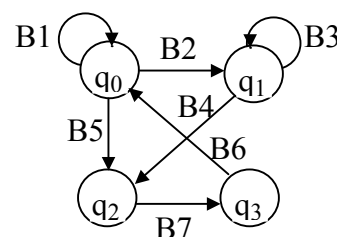


Рис. 3

- Сети Петри (рис. 4). Сети Петри являются направленными графами с двумя видами узлов, а именно с узлами для изображения состояний q_i (кружки) и узлами для изображения переходов (вертикальные штрихи) между состояниями. Переход осуществляется, если состояния, находящиеся перед символом перехода, помечены и наступает событие, вызывающее переход. Например, имеет место переход от q_0 к q_1 , q_2 и q_3 , если имеется q_0 и выполнено условие $B1$. Сети Петри особенно удобны для изображения параллельно происходящих взаимосвязанных процессов.

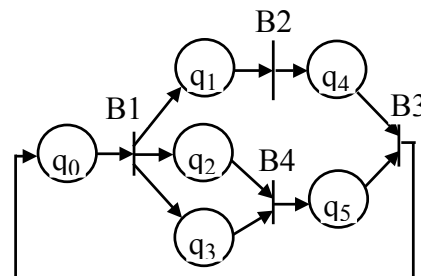


Рис. 4

- Графы последовательного выполнения программы. Пригодны для записи задач управления и для описания поведения релейных систем управления. Изображают зависящую от каких-либо условий последовательность состояний системы q_i . Положение (0 или 1) конкретных функциональных элементов ($Q1$, $Q2$, $Y1$, $Y2$, $Y3$), соответствующие некоторым характерным состояниям системы, указываются в отдельной таблице. В приведенном примере (рис. 5): как только $S1 = 1$, система совершает переход из состояния q_0 в q_1 ; если $S2 = 1$, то осуществляет-

ся переход в состояние q_2 , для $S2 = 0$ – в состояние q_0 и т. д.

- Схемы работы (рис. 6). Очень удобно использовать для описания линейно протекающих процессов и работы соответствующих систем управления. Изображается зависящая от появления определенных событий последовательность отдельных шагов или состояний процесса (q_0, q_1, q_2, \dots). Пример: сигнал "Пуск" и сигнал "закрыть заслонку" начинают первый шаг процесса (напри-

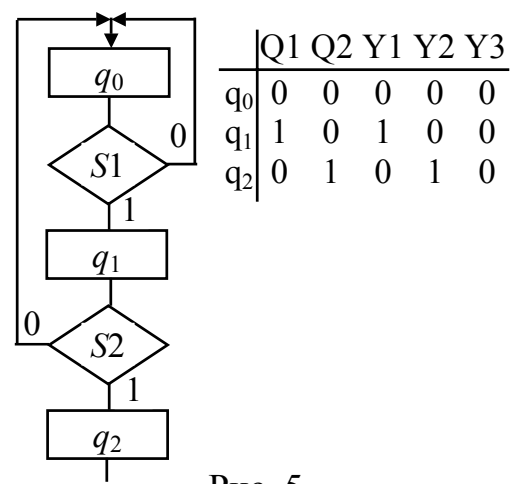


Рис. 5

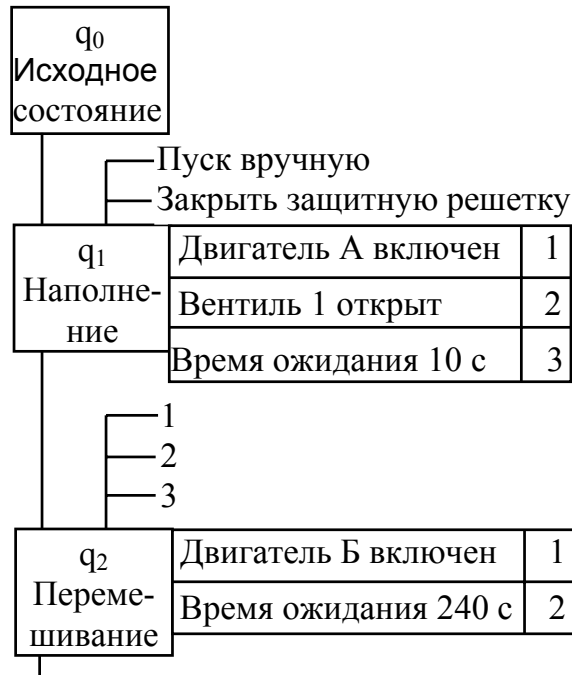


Рис. 6

мер, наполнения мешалки). Как только поступают информационные сигналы 1, 2, 3 (двигатель A включен, вентиль 1 открыт, время ожидания истекло), следует шаг процесса 2 и т. д.

1.4.3. Псевдокоды

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному, естественному языку, по-

этому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же как и

в формальных языках, есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примером псевдокода является школьный алгоритмический язык (АЯ) [1], содержащий систему обозначений для единообразной и точной записи алгоритмов и задания правил их использования. Важной особенностью алгоритмических языков типа псевдокодов является их близость к языкам программирования.

Как и любой язык, АЯ строится на основе алфавита, включающего в себя набор символов, разрешенных к использованию при написании алгоритмов.

Алфавит АЯ включает в себя строчные и прописные буквы русского и латинского алфавитов; цифры десятичной системы счисления; специальные символы, имеющиеся на клавиатуре устройства ввода данных ПК и в наборах устройств печати; символы математических операций, используемых при написании выражений.

Для дополнения символов алфавита в АЯ вводятся так называемые ключевые (служебные) слова, которые позволяют сделать запись алгоритма более понятной и выразительной. Они используются для формирования типовых синтаксических конструкций. Наборы ключевых слов алгоритмического языка типа АЯ приведен в табл. 2.

Таблица 2

Основные ключевые слова		Дополнительные ключевые слова
1		2
<u>алг</u> (алгоритм)	<u>рез</u> (результат)	<u>запись</u>
<u>нач</u> (начало)	<u>кон</u> (конец)	<u>истина</u>
<u>арг</u> (аргумент)	<u>знач</u> (значение)	<u>лог</u> (логический)
<u>тип</u>		<u>ложь</u>
<u>вещ</u> (вещественный)	<u>цел</u> (целый)	<u>массив</u>

1	2
<u>лит</u> (литерный) <u>таб</u> (таблица)	<u>множество</u>
<u>сим</u> (символьный)	<u>функция</u>
<u>не</u> <u>то</u> <u>если</u> <u>и</u>	<u>дано</u>
<u>все</u> <u>или</u> <u>выбор</u> <u>иначе</u>	<u>надо</u>
<u>нц</u> (начало цикла) <u>кц</u> (конец цикла)	<u>ввод</u>
<u>от</u> <u>до</u> <u>шаг</u> <u>для</u>	<u>вывод</u>
<u>пока</u> := (оператор присваивания)	<u>утв</u> (утверждение)
<u>при</u> <u>да</u> <u>нет</u>	

Алгоритм, записанный на алгоритмическом языке, начинается с заголовка и затем содержит последовательно расположенные друг за другом операторы (команды), которые образуют тело алгоритма. Тело алгоритма заключается между ключевыми словами нач и кон. Ниже приводится пример общего вида алгоритма:

алг название алгоритма (аргументы и результаты)

дано условия применимости алгоритма

надо цель выполнения алгоритма

нач описание промежуточных величин,
последовательность команд (тело алгоритма)

кон

Часть алгоритма от слова алг до слова нач называется **заголовком**, а часть, заключенная между словами нач и кон, – **телом** алгоритма.

Признаком заголовка алгоритма является ключевое слово алг, следом за которым указывается название алгоритма. Название должно отражать специфику решаемой задачи и лаконично записываться в виде отдельного предложения или некоторой аббревиатуры.

В предложении алг после названия алгоритма в круглых скобках указываются **характеристики** (арг, рез) и **тип значения** (цел, вещ, сим, лит или лог) всех входных (аргументы) и выходных (результаты) переменных.

Примеры предложений алг:

- 1) алг Объем и площадь цилиндра (арг вещ R, H , рез вещ V, S)
- 2) алг Корни КвУр (арг вещ a, b, c , рез вещ x_1, x_2 , рез лит t)
- 3) алг Исключить элемент (арг цел N , арг рез вещ таб $A[1:N]$)

Предложения **дано** и **надо** необязательны. В них рекомендуется записывать утверждения, описывающие состояние среды исполнителя алгоритма, например:

- 1) **алг** Замена (**арг лит** *Str1*, *Str2*, **арг рез лит** *Text*)
дано | длины подстрок *Str1* и *Str2* совпадают
надо | всюду в строке *Text* подстрока *Str1* заменена на *Str2*
- 2) **алг** Число максимумов (**арг цел** *N*, **арг вещ таб** *A*[1:*N*], **рез цел** *K*)
дано | $N > 0$
надо | *K* – число максимальных элементов в таблице *A*.

Здесь в предложениях **дано** и **надо** после знака "|" записаны комментарии. Комментарии можно помещать в конце любой строки. Они не обрабатываются транслятором, но существенно облегчают понимание алгоритма.

Для обозначения (именования) величин, а также других объектов; с которыми производится работа в алгоритмах, используются символические имена в виде идентификаторов.

Синтаксические конструкции языка подразделяются на два типа (см. рис. 7): описания данных (величин) и операторов (команд).

Описание данных производится путем отнесения их к одному из типов, принятому в алгоритмическом языке. Для АЯ такими типами данных являются целые, вещественные и литерные. К ним часто добавляются логические и натуральные типы значений.

Значения, представленные в виде констант того или иного типа, определяются по виду их написания. Так, целые числа записываются последовательностью цифр со знаком или без него: 0; -7; 23; +107; -1250; 2003.

Вещественные числа записываются в виде десятичной дроби, состоящей из целой и дробной частей, разделенных запятой, и содержащей при необходимости знак "–" или "+": –3,1415926; 135,18; –0,346; +13,4.

Литерные (символьные) значения – это отдельные символы алфавита алгоритмического языка или последовательности их, заключенные в кавычки (апострофы): 'g', s1, 'Q3', 'A12', 'ROOT'.

Логических значений всего два – истина и ложь.

Натуральные числа являются подмножеством целых. Следует иметь в виду, что число нуль (0) не входит в подмножество натуральных чисел.

Описание типа данных применяется для тех величин, которые используются в алгоритмах на языке АЯ. К ним относятся переменные и массивы, пред-

ставленные в виде одно- или двумерных таблиц.

Описание типа начинается с соответствующего ключевого слова и содержит список обозначений величин, которые относят к данному типу. При этом число описаний одного и того же типа не ограничивается.

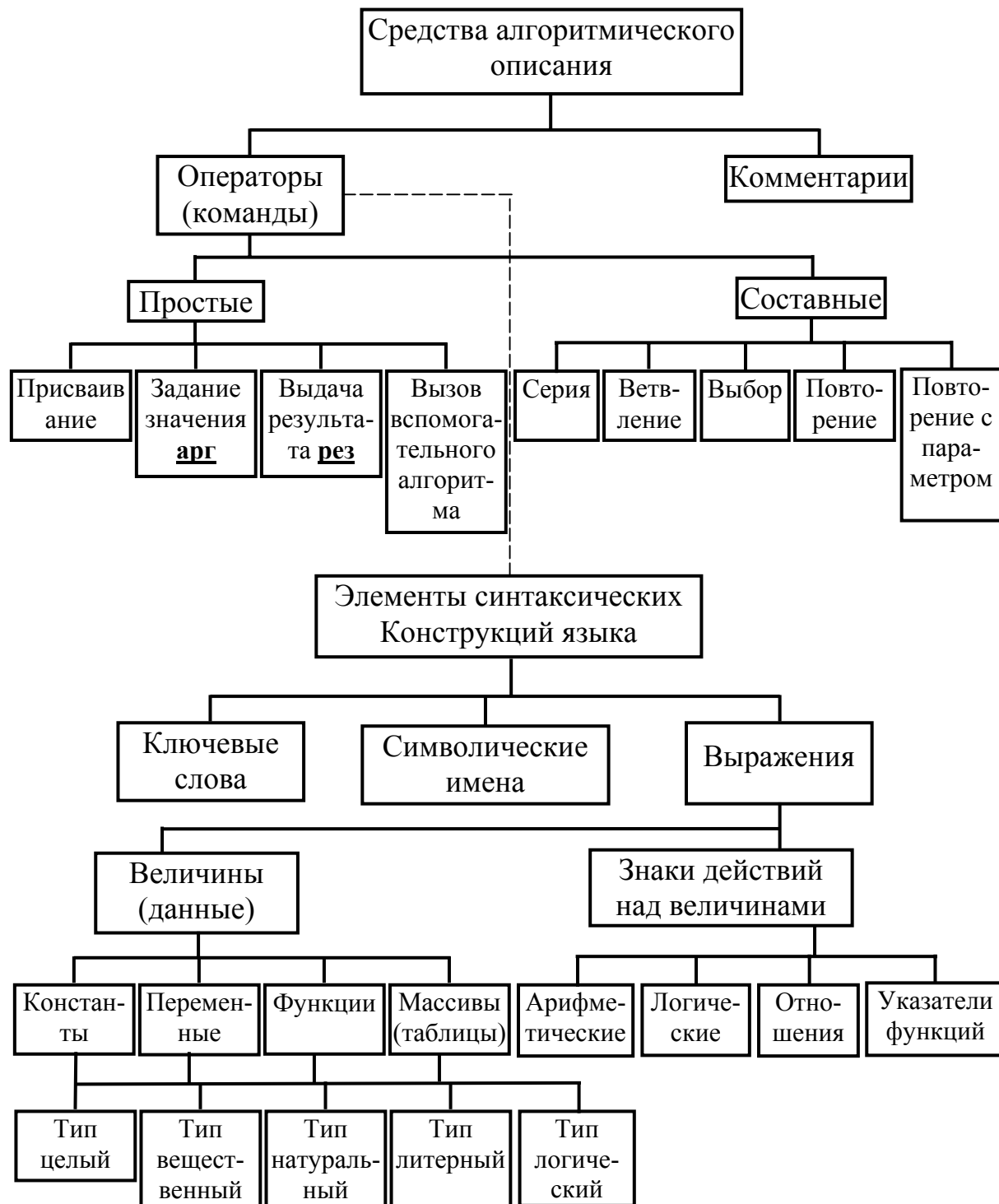


Рис. 7

Для описания переменных типа "целый" используется ключевое слово цел, типа "вещественный" – вещ, типа "литерный" – лит, типа

"символьный" – сим, типа "логический" – лог. Список описаний может быть задан как: цел k, m ; вещ a, z, c ; лит p, q ; сим i, j ; лог s, t .

Описания типов в списке могут быть заданы для каждой величины: цел k , цел h , цел l , вещ a , лог jt и т. д.

Характеристика типов значений массивов (таблиц) задается в виде составного описания, включающего в себя два ключевых слова, одно из которых определяет тип значения (например, вещ), а другое – тип применяемой величины таб.

Кроме того, при описании массивов (таблиц) обозначение используемой структуры данных – массива – задается ее идентификатором, дополненным списком граничных пар, задающим диапазон изменения индексов массива по каждому измерению (например, по строкам и столбцам матрицы). Пример описания типов массивов (таблиц): вещ табл $f[1:50]$; цел табл $d[1:5,1:20]$. Таблицы могут иметь и переменные границы: вещ табл $k[n:m]$. На момент начала работы с таблицей ее границы должны быть четко определены.

В алгоритм можно вводить и переменные какого-либо типа, не совпадающие ни с одним из стандартных. Такой тип задается перечислением значений, которые может принимать переменная. Общий вид описания нестандартного вида: тип $VN = (zn1, zn2, ..., znI)$, где VN – идентификатор типа, $zn1, zn2, ..., znI$ – конкретные значения, которые может принимать переменная типа VN . Например, тип FIGURA = (треугольник, квадрат, круг, пирамида).

Оператор присваивания является наиболее используемым при составлении алгоритмов. Общий вид оператора следующий: $Z := N$, где " $:=$ " есть символ (ключевое слово) оператора присваивания; N – выражение, значение которого определяется по данному оператору и присваивается величине Z . Выражение N может содержать константы, переменные, указатели функций, знаки операций и скобки. В зависимости от типов используемых в выражении N величин и знаков операций выражения могут быть арифметическими или логическими. Вид выражения однозначно задает правила определения его значения: действия выполняются слева направо с соблюдением следующего старшинства (в порядке убывания):

- 1) не (логическая операция отрицания);
- 2) $\cdot, /, \div, \text{и}$ (операция логического умножения);
- 3) $+, -, \text{или}$ (операция логического сложения);
- 4) $=, \neq, <, >, \leq, \geq$ (операции отношений).

Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам.

Присваивание допускается для величин всех типов, в том числе и массивов (таблиц).

В операторе $Z := N$ переменная Z и выражение N должны иметь один и тот же тип. Примеры операторов присваивания: $a := (b + c) \cdot \sin(\pi/4)$; $i := i + 1$.

В выражениях могут использоваться различные математические функции.

Оператор выдачи результата записывается в виде: **рез** список величин.

Такой оператор интерпретируется как команда, предписывающая после выполнения алгоритма вывод на какой-либо носитель информации (лист бумаги, экран дисплея) искомым результатов в виде значений величин, перечисленных в списке.

Следует заметить, что операторы **арг** и **рез** определены в АЯ слишком примитивно, они лишь обозначают необходимость выполнения действий по вводу и выводу значений данных, с которыми приходится иметь дело при выполнении алгоритма. Пример записи алгоритма:

алг площадь треугольника (**арг вещь** a, b , **рез вещь** s)

нач

вещ h

$h := a/2$

$s := h \cdot b$

кон

Для ввода и вывода данных используют команды:

- **ввод** имена переменных

- **вывод** имена переменных, выражения, тексты.

Для ветвления применяют команды **если** и **выбор**, для организации циклов – команды **для** и **пока**.

Рассмотрим пример записи алгоритма на АЯ:

алг Сумма квадратов (**арг цел** n , **рез цел** S)

дано | $n > 0$

надо | $S = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 + \dots + n \cdot n$

нач

цел i

ввод n ; $S := 0$

нц для i **от** 1 **до** n

$$S := S + i \cdot i$$

КЦ

ВЫВОД "S = ", S

КОН

1.4.4. Программный способ представления алгоритмов

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются компьютеры или иные вычислительные устройства (однокристальные микроПК, промышленные компьютеры, технологические контроллеры и др.). Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на понятном ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке – программой для компьютера.

К алгоритмическим языкам относят машинный язык (система команд), языки программирования.

Математическое обеспечение – средства, которые могут быть предоставлены пользователю для решения его задачи с помощью ПК. Оно включает в себя алгоритмическое обеспечение – методы и алгоритмы, модели решения задач, лингвистическое обеспечение – языки программирования, программное обеспечение – систему автоматизации программирования и информационное обеспечение – структуры данных и базы данных.

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм, как мы знаем, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости от степени детализации предписаний обычно определяется уровень языка программирования – чем меньше детализация, тем выше

уровень языка.

По этому критерию можно выделить следующие уровни языков программирования: машинные; машинно-ориентированные (языки ассемблера); машинно-независимые (языки высокого уровня).

Машинные и машинно-ориентированные языки – это языки низкого уровня, требующие указания мелких деталей процесса обработки данных. Языки же высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Языки высокого уровня делятся на:

- процедурные (алгоритмические) (Basic, Pascal, C и др.), которые предназначены для однозначного описания алгоритмов; для решения задачи процедурные языки требуют в той или иной форме явно выписать процедуру ее решения;
- логические (Prolog, Lisp и др.), которые ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания;
- объектно-ориентированные (Object Pascal, C++, Java и др.), в основе которых лежит понятие объекта, сочетающего в себе данные и действия над ними. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.

1.4.4.1. Достоинства и недостатки машинных языков

Каждый компьютер имеет свой машинный язык, т. е. свою совокупность машинных команд, которая отличается количеством адресов в команде, назначением информации, задаваемой в адресах, набором операций, которые может выполнить машина, и др. При программировании на машинном языке программист может держать под своим контролем каждую команду и каждую ячейку памяти, использовать все возможности имеющихся машинных операций.

Но процесс написания программы на машинном языке очень трудоемкий и утомительный. Программа получается громоздкой, труднообозримой, ее трудно

отлаживать, изменять и развивать.

Поэтому в случае, когда нужно иметь эффективную программу, в максимальной степени учитывающую специфику конкретного компьютера, вместо машинных языков используют близкие к ним машинно-ориентированные языки (ассемблер).

Основные преимущества алгоритмических языков программирования перед машинными и машинно-ориентированными языками: алфавит алгоритмического языка значительно шире алфавита машинного языка, что существенно повышает наглядность текста программы; набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса; формат предложений достаточно гибок и удобен для использования, что позволяет с помощью одного предложения задать достаточно содержательный этап обработки данных; требуемые операции задаются с помощью общепринятых математических обозначений; данным в алгоритмических языках присваиваются индивидуальные имена, выбираемые программистом; в языке может быть предусмотрен значительно более широкий набор типов данных по сравнению с набором машинных типов данных.

Алгоритмические языки программирования в значительной мере являются машинно-независимыми. Они облегчают работу программиста и повышают надежность создаваемых программ.

1.4.4.2. Компоненты алгоритмического языка

Алгоритмический язык (как и любой другой язык) образуют три составляющие: алфавит, синтаксис и семантика.

Алфавит – это фиксированный для данного языка набор основных символов, т. е. "букв алфавита", из которых должен состоять любой текст на этом языке, – никакие другие символы в тексте не допускаются.

Синтаксис – это правила построения фраз, позволяющие определить, правильно или неправильно написана та или иная фраза. Точнее говоря, синтаксис языка представляет собой набор правил, устанавливающих, какие комбинации символов являются осмысленными предложениями на этом языке.

Семантика определяет смысловое значение предложений языка. Являясь системой правил истолкования отдельных языковых конструкций, семантика устанавливает, какие последовательности действий описываются теми или

иными фразами языка и в конечном итоге какой алгоритм определен данным текстом на алгоритмическом языке.

1.4.4.3. Понятия, используемые в алгоритмических языках

Каждое понятие алгоритмического языка подразумевает некоторую синтаксическую единицу (конструкцию) и определяемые ею свойства программных объектов или процесса обработки данных. Понятие языка определяется во взаимодействии синтаксических и семантических правил. Синтаксические правила показывают, как образуется данное понятие из других понятий и букв алфавита, а семантические правила определяют свойства данного понятия.

Основными понятиями в алгоритмических языках обычно являются следующие:

- Имена (идентификаторы) – употребляются для обозначения объектов программы (переменных, массивов, функций и др.).
- Операции бывают следующих типов:
 - арифметические операции $+$, $-$, \cdot , $/$ и др.;
 - логические операции **и**, **или**, **не**;
 - операции отношения $<$, $>$, \leq , \geq , $=$, \neq ;
 - операция сцепки (иначе присоединения, конкатенации) символьных значений друг с другом с образованием одной длинной строки; изображается знаком "+".

Данные – величины, обрабатываемые программой. Имеется четыре основных вида данных: константы, переменные, массивы и структуры данных (списки, стеки и очереди, деревья и леса). Константы – это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.

Переменные обозначаются именами и могут изменять свои значения в ходе выполнения программы. Переменные бывают целые, вещественные, логические, символьные и литерные.

Массивы – последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним – в случае одномерного массива или несколькими – если массив многомерный). Иногда массивы называют таблицами.

Линейный список представляет собой способ организации последователь-

ности однотипных элементов, при котором каждый элемент, кроме первого, имеет одного предшественника (предыдущий элемент) и каждый элемент, кроме последнего, имеет одного преемника (следующий элемент). Доступ к каждому элементу списка можно получить, последовательно продвигаясь по списку от элемента к элементу. Другие типы списков, а также стеки, очереди, деревья и леса подробно рассмотрены в [5, 17].

Выражения предназначены для выполнения необходимых вычислений, состоят из констант, переменных, указателей функций (например, $\sin(x)$), объединенных знаками операций. Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, "многоэтажных" дробей и т. д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры. Различают выражения арифметические, логические и строковые.

Арифметические выражения служат для определения одного числового значения. Например, $(1 + \sin(x))/2$. Значение этого выражения при $x = 0$ равно 0.5, а при $x = \pi/2$ – единице.

Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения – "истина" или "ложь" (да или нет).

Значения строковых (литерных) выражений – тексты. В них могут входить литерные константы, литерные переменные и литерные функции, разделенные знаком операции сцепки.

Операторы (команды). Оператор – это наиболее крупное и содержательное понятие языка: каждый оператор представляет собой законченную фразу языка и определяет некоторый, вполне законченный этап обработки данных. В состав операторов входят: ключевые слова; данные; выражения и т. д.

Операторы подразделяются на исполняемые и неисполняемые. Неисполняемые операторы предназначены для описания данных и структуры программы, а исполняемые – для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и др.).

При решении различных задач с помощью компьютера бывает необходимо вычислить логарифм или модуль числа, синус угла и т. д. Вычисления часто употребляемых функций осуществляются посредством подпрограмм, называемых стандартными функциями, которые заранее запрограммированы и встроены в

транслятор языка. В качестве аргументов функций можно использовать константы, переменные и выражения. Каждый язык программирования имеет свой набор стандартных функций.

Арифметические выражения записываются по следующим правилам: 1. нельзя опускать знак умножения между множителями и ставить рядом два знака операций; 2. индексы элементов массивов записываются в квадратных (C++, Pascal) или круглых (Basic) скобках; 3. для обозначения переменных используются буквы латинского алфавита; 4. операции выполняются в порядке старшинства: сначала вычисление функций, затем возведение в степень, потом умножение и деление и в последнюю очередь сложение и вычитание. Операции одного старшинства выполняются слева направо.

В записи логических выражений, помимо арифметических операций сложения, вычитания, умножения, деления и возведения в степень, используются операции отношения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), = (равно), <> (не равно), а также логические операции – и, или, не.

К основным принципам написания программ относятся: 1) соблюдение требований структурного программирования (модульность, наличие лишь одной точки входа и одной точки выхода в процедурах, максимально возможный отказ от оператора go to); 2) использование комментариев для описания названия и назначения программ и отдельных модулей, идентификаторов и т.д.; 3) ступенчатое расположение операторов программы, визуальное выделение отдельных процедур, функций, внутренних и внешних циклов и т.д.

В качестве примера рассмотрим запись программы на языке C++ для нахождения наибольшего элемента заданной матрицы $B[1:10, 1:10]$.

```
#include <iostream.h>      // подключение заголовочного файла
#define n 10               // nхn- размер массива
int i, max, B[n][n];       // max - максимальное значение элемента
void main( )
{
    cout << "Ввод массива B\n";
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            {cout << "B[" << i << ", " << j << "]= " ; cin >> B[i][j];}
    max = B[0][0];          // инициализация переменных
    for(i = 1; i <= n; i++)
        for (j = 0; j < n; j++)
```

```

        if(B[i][j] > max)
            max = B[i][j]; // переопределение максимума
    cout << "max = " << max; // вывод максимального элемента
}

```

1.5. Структуры алгоритмов

Преобразования величин, реализуемые в алгоритмическом языке, осуществляются по операторам (командам), располагаемым в заданной последовательности. Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следование, ветвление, цикл. Характерной особенностью базовых структур является наличие в них одного входа и одного выхода.

Структура алгоритма является линейной, если она образована последовательностью простых операторов (команд).

Разветвляющийся алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого ПК обеспечивает переход на один из двух возможных шагов.

Циклический алгоритм – алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. Группа команд (операторов), выполняющихся одна за другой, называется серией. Серия может состоять из одного оператора.

Для построения разветвляющихся и циклических структур алгоритма в алгоритмическом языке используются составные операторы. К ним относятся операторы ветвления и цикла.

Оператор ветвления записывается следующим образом: **если** условие **то** серия 1 **иначе** серия 2 **все**. В зависимости от итога проверки условия выполняется только одна из двух серий, входящих в команду ветвления. Если условие соблюдено, то следует выполнять серию 1, если нет – серию 2. Оператор ветвления используется и в сокращенной форме: **если** условие **то** серия **все**

При этом, если условие соблюдено, необходимо выполнить серию команд, следующую в записи алгоритма за служебным словом **то**, в противном случае, пропуская серию, перейти к выполнению команды, следующей за командой ветвления (после служебного слова **все**).

Структура ветвление существует в четырех основных вариантах [1] (см. табл. 3): **если – то**; **если – то – иначе**; **выбор**; **выбор – иначе**.

Оператор выбора используется в тех случаях, когда возникает необходи-

мость выбора альтернативы из трех возможностей и более.

Различия в исполнении этих конструкций вытекают из свойств ветвления.

С формальной точки зрения рассмотренные конструкции эквивалентны, их использование определяется удобством составления алгоритма.

Оператор повторения (цикла) используется для описания алгоритмов, в которых требуется организовать многократное повторение одних и тех же действий.

Циклические структуры (см. табл. 3) имеют особое значение для построения алгоритмов, так как только на их основе можно добиться компактной записи алгоритмов, требующих выполнения большого числа действий.

При выполнении этого оператора серия, включающая одну или несколько команд, повторяется несколько раз подряд до тех пор, пока условие соблюдается. Как только условие нарушается, выполнение серии прекращается. Если условие изначально неверно, то серия не выполняется.

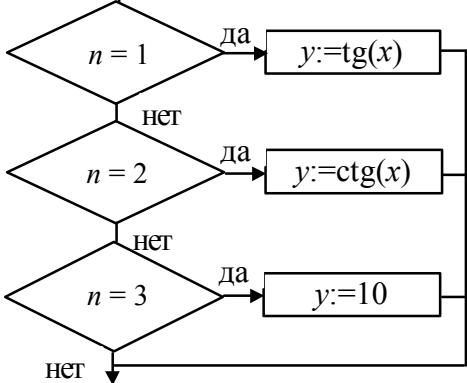
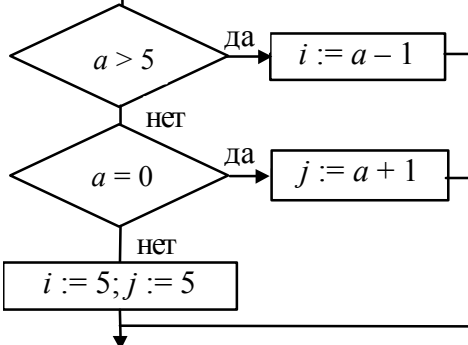
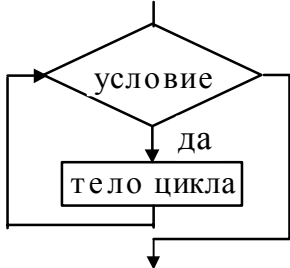
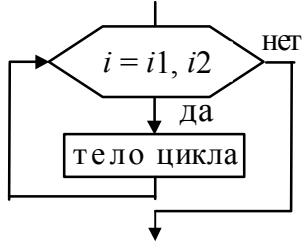
Алгоритм, в состав которого входит итерационный цикл, называется итерационным алгоритмом. Итерационные алгоритмы используются при реализации итерационных численных методов.

Возможны случаи, когда внутри тела цикла необходимо повторить некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название цикла в цикле или вложенных циклов. Глубина вложения циклов (количество вложенных друг в друга циклов) может быть различной.

Таблица 3

Алгоритмический язык	Схема алгоритма
<u>если - то</u>	
<u>если</u> условие <u>то</u> действия <u>все</u>	
<u>если - то - иначе</u>	
<u>если</u> условие <u>то</u> действия 1 <u>иначе</u> действия 2 <u>все</u>	

<u>выбор</u>	
<p><u>выбор</u></p> <p><u>при</u> условие 1: действия 1</p> <p><u>при</u> условие 2: действия 2</p> <p>...</p> <p><u>при</u> условие n: действия n</p> <p><u>все</u></p>	<pre> graph TD Start(()) --> D1{условие 1} D1 -- да --> A1[действия 1] D1 -- нет --> D2{условие 2} D2 -- да --> A2[действия 2] D2 -- нет --> Dn{условие n} Dn -- да --> An[действия n] Dn -- нет --> Exit(()) A1 --> Exit A2 --> Exit An --> Exit </pre>
<u>выбор – иначе</u>	
<p><u>выбор</u></p> <p><u>при</u> условие 1: действия 1</p> <p><u>при</u> условие 2: действия 2</p> <p>...</p> <p><u>при</u> условие n: действия n</p> <p><u>иначе</u> действия $n + 1$</p> <p><u>все</u></p>	<pre> graph TD Start(()) --> D1{условие 1} D1 -- да --> A1[действия 1] D1 -- нет --> D2{условие 2} D2 -- да --> A2[действия 2] D2 -- нет --> Dn{условие n} Dn -- да --> An[действия n] Dn -- нет --> An1[действия n+1] A1 --> Exit(()) A2 --> Exit An --> Exit An1 --> Exit </pre>
Примеры команды <u>если</u>	
<p><u>если</u> $x > 0$</p> <p><u>то</u> $y := \text{tg}(x)$</p> <p><u>все</u></p>	<pre> graph TD Start(()) --> D1{x > 0} D1 -- да --> A1[y:=tg(x)] D1 -- нет --> Exit(()) A1 --> Exit </pre>
<p><u>если</u> $a > b$</p> <p><u>то</u> $a := 5 \cdot a; b := 1$</p> <p><u>иначе</u> $b := 4 \cdot b + 1$</p> <p><u>все</u></p>	<pre> graph TD Start(()) --> D1{a > b} D1 -- да --> A1[a:=a*5; b:=1] D1 -- нет --> A2[b:=4*b+1] A1 --> Exit(()) A2 --> Exit </pre>

<p><u>выбор</u></p> <p><u>при</u> $n = 1$: $y := \text{tg}(x)$</p> <p><u>при</u> $n = 2$: $y := \text{ctg}(x)$</p> <p><u>при</u> $n = 3$: $y := 10$</p> <p><u>все</u></p>	
<p><u>выбор</u></p> <p><u>при</u> $a > 5$: $i := a - 1$</p> <p><u>при</u> $a = 0$: $j := a + 1$</p> <p><u>иначе</u> $i := 5; j := 5$</p> <p><u>все</u></p>	
<p align="center">Цикл типа <u>пока</u></p> <p>Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова <u>пока</u>.</p>	
<p><u>нц пока</u> условие</p> <p>тело цикла (последовательность действий)</p> <p><u>кц</u></p>	
<p align="center">Цикл типа <u>для</u></p> <p>Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.</p>	
<p><u>нц для</u> i <u>от</u> $i1$ <u>до</u> $i2$</p> <p>тело цикла (последовательность действий)</p> <p><u>кц</u></p>	

Примеры команд <u>пока</u> и <u>для</u>	
$i := 1; S := 0$ <u>нц пока</u> $i \leq 10$ $S := S + A[i]$ $i := i + 1$ <u>кц</u>	<pre> graph TD Entry(()) --> Cond{i <= 10} Cond -- да --> P1[S := S + A[i]] P1 --> P2[i := i + 1] P2 --> Entry Cond -- нет --> Exit(()) </pre>
<u>нц для</u> i <u>от</u> 1 <u>до</u> 8 $X[i] := i \cdot i$ $Y[i] := X[i]/8$ <u>кц</u>	<pre> graph TD Entry(()) --> Cond{i = 1, 8} Cond -- да --> P1[X[i] := i * i] P1 --> P2[Y[i] := X[i] / 8] P2 --> Entry Cond -- нет --> Exit(()) </pre>

При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла вложенных циклов для.

Рассмотрим несколько примеров на циклические структуры.

Пример 1 (Организация цикла). Составить алгоритм вычисления суммы s вещественных чисел, образующих таблицу A из 10 элементов, пронумерованных от 1 до 10. Для подсчета просуммированных элементов используется промежуточная целая переменная i :

Вариант 1:

алг Сумма вещественных чисел (**арг вещ таб** $A[1:10]$, **рез вещ** s)

нач цел i

$s := 0; i := 1$ | начальные значения

нц пока $i \leq 10$ | условие выхода из цикла

ввод A

$s := s + A[i]$ | расчет суммы вещественных чисел таблицы A

кц

вывод s

кон

Вариант 2:

алг Сумма вещественных чисел (**арг вещ таб** $A[1:10]$, **рез вещ** s)

нач цел i

$s := 0$ | начальное значение суммы

нц для i **от** 1 **до** 10 | начало цикла по расчету суммы

ВВОД A

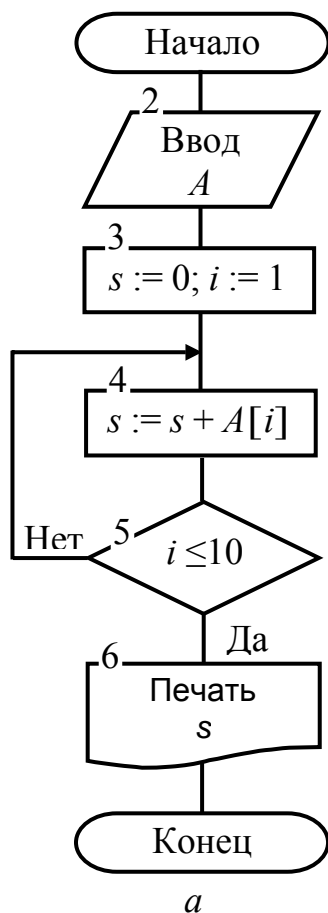
$s := s + A[i]$ | расчет суммы вещественных чисел таблицы A

КЦ

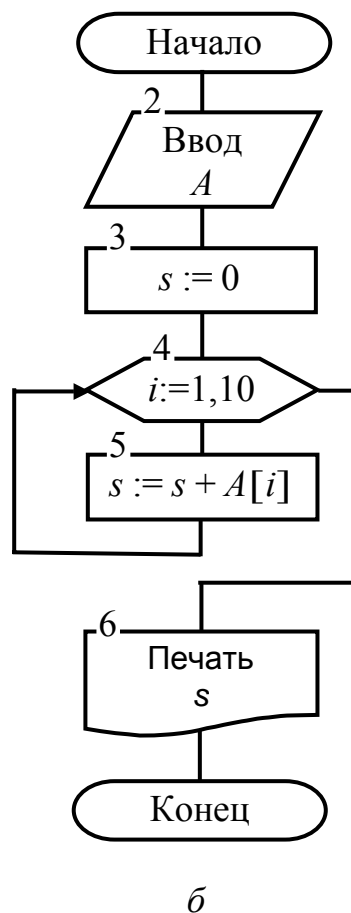
ВЫВОД s

КОН

На рис. 8, *а*, *б* показаны 2-а варианта графического представления алгоритма:



а



б

Рис. 8

Приведенный пример содержит алгоритм поэлементной обработки линейной таблицы, характерный для многих применений. Для общего случая линейной таблицы **таб** $A[N:K]$ его можно представить в виде:

вариант 1

$I := N$

пока $I \leq K$

нц

 обработка $A[I]$

$I := I + 1$

кц

вариант 2

$I := N$

нц **для** I **от** N **до** K

 обработка $A[I]$

кц

Анализ этого алгоритма показывает, что в общем случае в подобных конструкциях используются: целая переменная (I), указывающая текущий элемент таблицы; начальное значение этой переменной (N), присваиваемое ей до начала

цикла; конечное значение (K), по достижении которого происходит выход из цикла; шаг переменной – изменение ее значения при каждом проходе цикла (в данном случае шаг равен 1).

Пример 2 (Вложенные циклы). Составить алгоритм нахождения наибольшего элемента заданной матрицы $B[1:10, 1:10]$. Для нахождения наибольшего элемента используются промежуточные целые переменные i, j :

алг Нахождение наибольшего элемента (**арг цел таб** $B[1:10, 1:10]$, **рез цел** Max)

ввод B

нач цел i, j

$Max := B[1, 1]$

для i **от** 1 **до** 10

нц

для j **от** 1 **до** 10

нц

если $B[i, j] > Max$

то $Max := B[i, j]$

все

кц

кц

вывод s

кон

На рис. 9 показано графическое представление алгоритма:

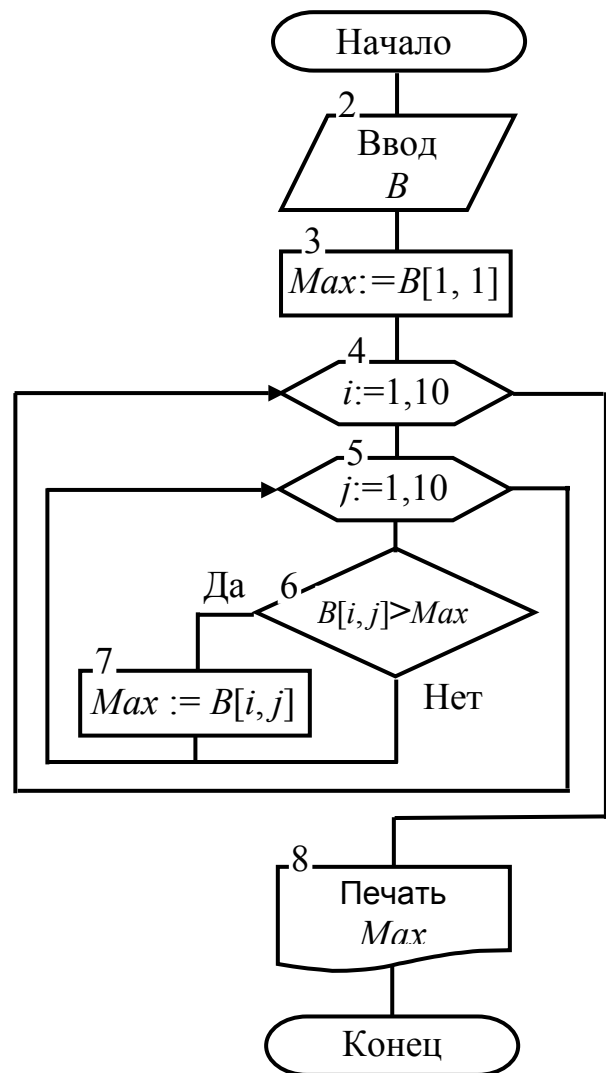


Рис. 9

Пример 3 (итерационный цикл). Составить алгоритм вычисления суммы членов бесконечного ряда с точностью ε

$$s = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + \dots$$

В качестве начального значения суммы следует взять значение первого слагаемого, равное 1. Для вычисления значения текущего члена ряда h целесообразно использовать прием накопления произведения, так как последующий член ряда можно вычислить, зная предыдущий, с помощью рекуррентной формулы $h_n = h_{n-1} \cdot x/n$. Поэтому начальное значение h следует взять равным единице.

На алгоритмическом языке алгоритм можно записать в следующем виде:

алг Сумма бесконечного ряда (**арг вещ** x , eps , **рез вещ** s)

дано | $0 < x < 1$

надо | $s = 1 + x + x^2/2! + \dots + x^n/n! + \dots$

нач цел n вещ h

ВВОД x, eps

$s := 1; h := 1; n := 1$ | начальные значения

нц пока $abs(h) \leq eps$

$h := h \cdot x / n$ | очередное слагаемое

$s := s + h$ | частичная сумма

$n := n + 1$ | номер очередного слагаемого

кц

ВЫВОД s

кон

А графическое представление алгоритма показано на рис. 10.

1.6. Вспомогательные алгоритмы

При построении новых алгоритмов могут использоваться алгоритмы, составленные ранее. Алгоритмы, целиком используемые в составе других алгоритмов, называют вспомогательными (или подчиненными) алгоритмами. Возможны случаи, когда алгоритм, содержащий ссылку на вспомогательный, в определенной ситуации также может оказаться в роли вспомогательного алгоритма.

В некоторых случаях при наличии одинаковых последовательностей указаний (команд) для различных данных с целью сокращения записи также выделяют вспомогательный алгоритм.

Вспомогательные алгоритмы описываются отдельно, в терминах так называемых формальных параметров, которые представлены обозначениями и характеризуют требуемый порядок выполнения алгоритма. Например, алгоритм поиска большего из двух чисел α и β (обозначим БИД) может быть записан следующим образом:

алг БИД (**арг вещ** α, β , **рез вещ** γ)

нач если $\alpha \geq \beta$

то $\gamma := \alpha$

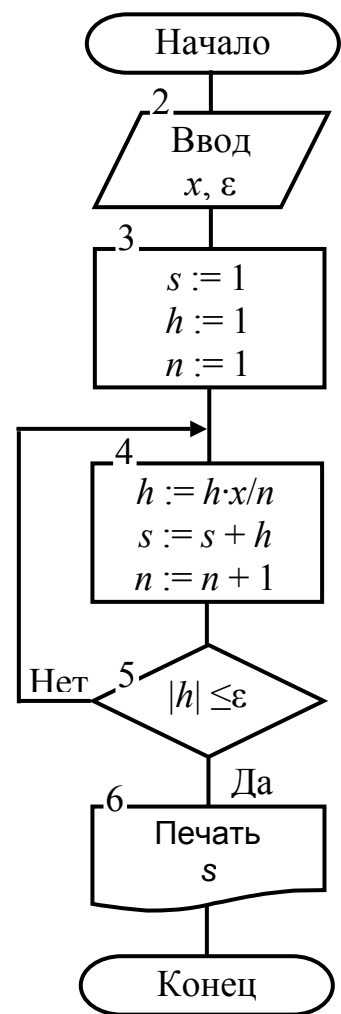


Рис. 10

иначе $\gamma := \beta$

все

кон

Обращение к вспомогательному алгоритму организуется с помощью специального оператора вызова, содержащего имя вспомогательного алгоритма и список, фактических параметров. При этом число, тип и порядок перечисления фактических параметров в списке должны полностью соответствовать числу, типу и порядку перечисления формальных параметров в заголовке вспомогательного алгоритма. Так, вызов вспомогательного алгоритма для поиска большего из трех чисел (обозначим его БИД) можно составить в форме двукратного обращения к алгоритму БИД:

алг БИД (**арг вещь** a, b, c , **рез вещь** y)

нач вещь z

БИД (a, b, z)

БИД (z, c, y)

кон

Фактические параметры a, b, c, y полностью соответствуют по смыслу формальным параметрам α, β, γ . Поэтому после исполнения вспомогательных алгоритмов в основном алгоритме можно воспользоваться переменной y как результатом наибольшего из трех чисел.

Наряду со стандартными функциями $\sin(x)$, $\cos(x)$, $\ln(x)$ и т. п., значения которых вычисляются по известным правилам, для определения значений других функций в АЯ вводится особый класс вспомогательных алгоритмов, которые выделяются и оформляются специальным образом.

В виде функций могут быть оформлены алгоритмы, имеющие в качестве результата одно значение простого (т. е. не табличного) типа. Функции различаются оформлением и способом использования. Обращение к функции не требует применения специального оператора. Оно осуществляется посредством указателя функции, тип которой характеризуется типом результата функции.

Указатель функции является элементом арифметического или логического выражения. В указателе функции записывается имя вспомогательного алгоритма и список фактических параметров, передаваемых в качестве аргументов: $\text{Integfun}(a, b, h)$, $\text{plosht}(a, b, c)$.

По указателю происходит обращение к функции, параметры-аргументы передаются по общим правилам, а полученный результат в виде одного значе-

ния возвращается в основной алгоритм как значение указателя функции и участвует в дальнейшем вычислении значения выражения. Так, выполнение оператора: $y := \text{интегфун}(a, b, h)$ приведет к обращению к вспомогательному алгоритму вычисления интеграла функции, а выполнение оператора: $s := \text{plosht}(a, b, c)$ приведет к обращению к вспомогательному алгоритму вычисления площади треугольника, при этом результатом обращения будут значения функций, соответствующие заданным фактическим аргументам.

Функции можно представить только как вспомогательные алгоритмы. Объясняется это тем, что результат функции рассматривается как одно из промежуточных значений, необходимых для вычисления выражения. Чтобы воспользоваться этим результатом, функцию необходимо не просто исполнить, а вызвать из какого-то алгоритма.

Особенности оформления функций заключаются в описании параметров. В список формальных параметров функций включаются только аргументы. Тип результата, получаемого функцией, указывается в заголовке перед именем вспомогательного алгоритма аналогично тому, как если бы это был тип алгоритма в целом. Тем самым вызов функции заменяет обращение к переменной соответствующего типа. Однако следует четко понимать, что тип относится не к самой функции, а к ее результату.

Примером описания вспомогательного алгоритма-функции является алгоритм вычисления площади треугольника по трем его сторонам:

вещ plosht(**арг вещ** a, b, c , **рез вещ** plosht)

нач вещ p

$$p := (a + b + c)/2$$

$$\text{plosht} := \text{sqrt}(p \cdot (p - a) \cdot (p - b) \cdot (p - c))$$

кон

Имя результата с точки зрения вызывающего алгоритма совпадает с именем функции. Поэтому может показаться естественным использование имени функции в качестве имени результата и в самой функции. Однако при этом становится невозможным рекурсивное обращение функции к самой себе, так как это обращение будет восприниматься как использование текущего значения результата. Поэтому в качестве имени результата при рекурсивных вызовах в АЯ используется специальное ключевое слово **знач**.

1.7. Технология решения задач с использованием компьютера

1.7.1. Основные этапы решения задач с помощью компьютера

Создание программы для ПК – сложный и трудоемкий процесс. Он включает в себя следующие основные этапы, показывающие логическую последовательность действий от постановки задачи до получения решения.

1. Общая формулировка задачи. Этот пункт, несмотря на кажущуюся простоту, чрезвычайно важен. Здесь необходимо сформулировать задачу в содержательных терминах и определить, что является "входными" данными задачи и что мы собираемся получить в результате решения. Недопустимо требование "найти то, сам не знаю что".

2. Математическая формулировка задачи. Здесь необходимо определить математические величины, которые будут описывать задачу, и получить математические связи между ними, т.е. составить математическую модель. Этот этап является критическим, поскольку неправильная или плохая модель сводит на нет все дальнейшие усилия. В то же время во многих случаях этот этап является очевидным, если есть общепринятые уравнения, описывающие рассматриваемый класс задач.

3. Выбор математического метода решения. Здесь необходимо на основе накопленного арсенала математических методов выбрать тот, который целесообразно использовать для решения поставленной задачи. Как правило, этот выбор осуществляется исходя как из субъективных причин (знание тех или иных математических методов), так и объективных причин, к которым в первую очередь необходимо отнести имеющиеся ресурсы компьютера (память, быстродействие). При этом если для получения решения требуются ресурсы, которые превосходят имеющиеся в наличии (чрезмерное время счета или недоступный объем памяти), то необходим поиск других математических методов, либо упрощение математической модели.

4. Составление алгоритма решения. Этот этап тесно связан с предыдущим и должен быть направлен в первую очередь на разработку эффективных алгоритмов, т.е. таких, которые требуют наименьшего количества ресурсов компьютера для своей реализации.

5. Составление и отладка программы. Этот этап может быть весьма трудоемким, особенно для начинающих программистов. При отладке больших программ целесообразно использовать специальные программные

средства, облегчающие процесс нахождения ошибок.

6. Тестирование программы. На этом этапе, чтобы удостовериться в правильности работы алгоритма, решаются задачи с такими исходными данными, для которых известно достоверное решение, либо используются какие-то косвенные свидетельства. Так в ряде задач существует связь между исходными данными и результатами, например закон сохранения энергии, импульса и т.д.

7. Решение поставленной задачи и представление результатов. Здесь наиболее существенным является удобный и наглядный вывод результатов. Во многих случаях целесообразно использовать графические программные средства для визуализации полученных данных.

При решении конкретных задач некоторые из этих этапов могут исключаться самой постановкой задачи. Например, если требуется вычислить значение некоторой функции $y = f(x)$ при различных значениях аргумента x , то формула заданной функции является математической формулировкой задачи и при этом определяет метод вычислений.

Пример. Привести постановку задачи, метод вычисления, сценарий и алгоритм определения среднего арифметического N чисел.

Постановка задачи:

Определение среднего арифметического N чисел.

Дано: N – количество чисел,

x_1, x_2, \dots, x_n – числа

Требуется: s – среднее N чисел

Где: $sr = (x_1 + x_2 + \dots + x_n)/N$.

При: $N > 0$.

Метод решения:

$$\left\{ \begin{array}{l} S_0 = 0 \\ \left\{ \begin{array}{l} S_n = S_{n-1} + x_n \\ [n = 1, \dots, N] \end{array} \right. \\ sr = \frac{S_n}{N} \end{array} \right.$$

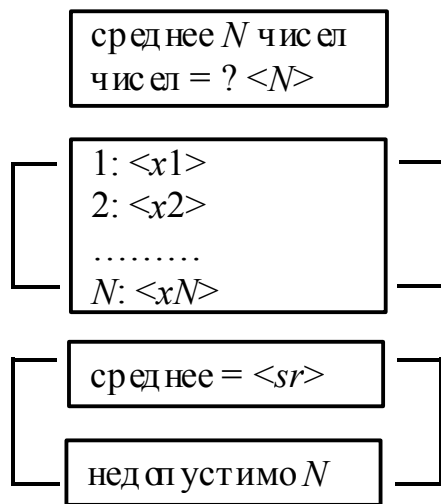
Сценарий:

Алгоритм решения:

алг Среднее арифметическое (**арг цел** N , **вещ** x , **рез вещ** sr)

нач цел n

$n := 1; S := 0$



```

ВВОД "Чисел = ", N
если  $N > 0$  то
    пока  $n \leq N$ 
        нц
            ВВОД x
             $S := S + x$ 
             $n := n + 1$ 
        кц
         $sr := S/N$ 
    вывод "Среднее N чисел =", sr
иначе
    вывод "Недопустимое значение N"
все
кон

```

Совокупность приемов, направленных на создание безошибочной программы за приемлемое время, составляет технологию программирования. К их числу относятся приемы разработки структуры программы, приемы включения в программу дополнительных операторов, обеспечивающих ускоренное создание программы и как можно более раннее обнаружение ошибок, и приемы, обеспечивающие возможность использования созданной программы как промышленного изделия.

Для каждого из этапов создания и использования программы существуют определенные приемы обеспечения качества программы.

Высокое качество программы достигается в первую очередь за счет глубокой проработки схемы алгоритма. Это прежде всего безошибочность программы, уверенность программиста в том, что она не содержит ошибок, и уверенность пользователя в том, что она правильна.

Хотя при разработке программы невозможно избежать ошибок, рекомендуется придерживаться некоторых правил составления алгоритма:

1. Необходимо стремиться к наиболее полному изучению поставленной задачи при формулировке задачи, т. е. разработке математической модели. Такое изучение позволяет добиться ясной, предусматривающей множество логических взаимодействий объектов программы. Недостаточно глубокая проработка математической модели приводит к неправильным результатам решения задачи.

2. Проработка алгоритма решения задачи связана с возможно более полным учетом общих особенностей процесса вычислений на ПК.

3. При разработке алгоритма необходимо стремиться к максимальной простоте и понятности. Это относится как к содержательной стороне, так и к форме записи программы на языке программирования. Применение стандартных приемов структурного программирования делает алгоритм (программу) более ясной, хотя в некоторых случаях более громоздкой и менее эффективной. Структурное программирование опирается на основные принципы системного подхода: а) программа должна состояться мелкими шагами; б) размер шага определяется количеством решений, применяемых программистом на этом шаге; в) сложная задача должна разбиваться на достаточно простые, легко воспринимаемые части (блоки), каждая из которых имеет только один вход и один выход; г) логика алгоритма (программы) должна опираться на минимальное число достаточно простых базовых управляющих структур.

Структурированная программа представляет собой композицию из последовательных или вложенных друг в друга блоков с одним входом и одним выходом, причем размеры этих блоков могут доходить до уровня элементарных предложений языка программирования (операторов).

1.7.2. Приемы алгоритмизации расчетных задач

Существует большое количество всевозможных приемов и методов разработки алгоритмов. Однако из всего многообразия методов разработки алгоритмов можно выделить небольшой набор основных, которые часто лежат в основе создания всевозможных алгоритмов и процедур, некоторые из них рассматриваются далее. Более подробно приемы и методы разработки алгоритмов описаны в [3].

1.7.2.1. Метод частных целей

Это метод сводится к тому, что сложная задача сводится к последовательности более простых задач. С другой стороны, в конкретной сложной задаче часто очень трудно указать способ ее разбиения на набор более простых задач. В этом случае большое значение имеет опыт и искусство программиста. Тем не менее, несмотря на общность метода и отсутствие "точного рецепта" его применение очень важно освоить этот метод, так как он лежит в основе решения многих задач и по своей сути составляет основу алгоритмизации и программирования.

В этом случае методика разработки алгоритма сводится к следующим операциям (действиям):

1. рассматривают вопрос о возможности разбиения задачи на последовательность более простых задач;
2. устанавливают взаимосвязь между сформированной последовательностью простых задач по средством переменных;
3. если необходимо, то на переменные вводятся допущения и ограничения;
4. составляется общий алгоритм решения поставленной задачи.

1.7.2.2. Метод подъема

Этот метод относится к одному из общих методов разработки алгоритмов. Алгоритм начинается с принятия начального предположения или построения начального решения задачи. Затем начинается (насколько возможно) быстрое движение "вверх" от начального уровня по направлению к лучшим решениям. Когда алгоритм достигает точки, из которой больше невозможно двигаться "наверх", он останавливается.

1.7.2.3. Программирование с отходом назад

Основой программ программирования игр, выбора решений, распознавания образов и т.п., – является программирование перебора вариантов. Программирование перебора вариантов – это сложная задача, так как алгоритмы перебора ищут решения не по заданным правилам вычислений, а путем проб и ошибок, и схема не укладывается в схемы циклов, имеющихся в языках программирования. Ситуация зачастую осложняется тем, что прямыми методами перебор всех возможных вариантов невозможно осуществить из-за их огромного количества.

Метод программирования с отходом назад позволяет осуществить организованный исчерпывающий поиск требуемого решения задачи. При этом часто удается избежать перебора всех возможных вариантов.

1.7.2.4. Алгоритмы ветвей и границ

Алгоритмы ветвей и границ применяются для решения переборных задач. Как и алгоритм с отходами, они исследуют древовидную модель пространства решений и ориентированы на поиск в некотором смысле оптимального решения (из конечного множества возможных решений-вариантов). В целях упрощения понимания сути алгоритмов такого рода рассмотрим одну конкретную задачу, в

которой такой алгоритм весьма хорошо работает и достаточно прост в понимании.

Вопросы для самопроверки

1. Дайте определение термину "алгоритм".
2. Назовите основные свойства алгоритмов, приведите примеры.
3. Назовите основные характеристики алгоритмов.
4. Назовите формы записи алгоритмов.
5. Назовите основные структуры алгоритмов.
6. Что понимается под разветвляющейся структурой алгоритма?
7. Что понимается под циклической структурой алгоритма?
8. Конструирование алгоритмов методом последовательной детализации.
9. Что понимается под термином "вспомогательный алгоритм".
10. Назовите этапы решения задач на компьютере.
11. Переменная в программировании: имя и значение.
12. Назовите основные приемы алгоритмизации расчетных задач.
13. Какие основные понятия используются в алгоритмических языках?
14. Перечислите средства алгоритмического описания.
15. Перечислите основные синтаксические конструкции алгоритмического языка.
16. Какие графические способы описания алгоритмов применяются в промышленных системах?
17. Что понимается под термином "Псевдокод".
18. В чем заключается программный способ представления алгоритмов?
19. Перечислите основные достоинства и недостатки машинных языков.
20. Какие виды данных Вы знаете?

Глава 2

ПРАКТИКУМ ПО АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЮ

2.1. Алгоритмы линейной структуры

При разработке алгоритма и написании программы необходимо помнить о том, что:

- алгоритмы и программы с линейной структурой являются простейшими и используются, как правило, для реализации простых вычислений по формулам;

- в них инструкции выполняются последовательно, одна за другой.

Пример 1. Составить алгоритм и написать программу вычисления объема цилиндра.

Схема алгоритма вычисления объема цилиндра:

алг Объем цилиндра (**арг вещ** r , h , **рез вещ** v)

нач

вещ $\pi = 3.1415926$

ввод r , h

$v = 2 * \pi * r * r * h$

вывод v

кон

Программа вычисления объема цилиндра:

```
// Вычисление объема цилиндра
#include <stdio.h>
#include <conio.h>
void main()
{
    float r, h, v, pi = 3.1415926; // радиус основания, высота
                                   // и объем цилиндра
    printf("Вычисление объема цилиндра \n");
    printf("Введите исходные данные: \n ");
    printf("Радиус основания (см)");
    scanf("%f", &r);
    printf("Высота цилиндра (см) ");
    scanf("%f", &h);
    v = 2*pi*r*r*h;
    printf("\nОбъем цилиндра %6.2f куб.см\n", v);
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}
```

Пример 2. Составить алгоритм и написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (км); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина.

Схема алгоритма вычисления стоимости поездки на автомобиле на дачу (туда и обратно) показана на рис. 11.

Программа вычисления стоимости поездки на автомобиле на дачу (туда и обратно):

```
// Вычисление стоимости поездки на дачу и обратно
#include <stdio.h>
#include <conio.h>
void main()
{
    float rast; // расстояние до дачи
    float potr; // потребление бензина на 100 км. пути
    float cena; // цена одного литра бензина
    float summ; // стоимость поездки на дачу и обратно
    printf("\nСтоимость поездки на дачу и обратно \n");
    printf("Расстояние до дачи (км) " );
    scanf("%f",&rast);
    printf("Расход бензина (литров на 100 км.) " );
    scanf("%f",&potr);
    printf("Цена литра бензина (руб.) " );
    scanf("%f",&cena);
    summ = 2 * potr/100 * rast * cena;
    printf("Поездка на дачу и обратно обойдется");
    printf("в %6.2f руб.", summ);
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}
```

Пример 3. Составить алгоритм и написать программу вычисления величины дохода по вкладу. Процентная ставка (% годовых) и время хранения (дней) задаются во время работы программы.

Схема алгоритма вычисления величины дохода по вкладу:

алг Величина дохода по вкладу (**арг вещ** *summ*, *stavka*, **арг int** *srok*,
вещ рез *dohod*, *summ*)

нач

ввод *summ*, *srok*, *stavka*

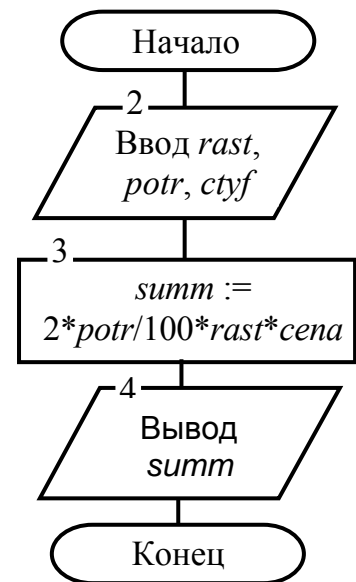


Рис. 11

$dohod = summ * stavka / 365 / 100 * srok$

$summ = summ + dohod$

ВЫВОД $dohod, summ$

КОН

Программа вычисления величины дохода по вкладу:

```
// Вычисление дохода по вкладу
#include <stdio.h>
#include <conio.h>
void main()
{
    float summ;      // сумма вклада
    int srok;         // срок вклада
    float stavka;     // процентная ставка
    float dohod;      // доход по вкладу
    printf("\nВычисление дохода по вкладу\n");
    printf("Введите исходные данные:\n");
    printf("Величина вклада (руб.) " );
    scanf("%f", &summ);
    printf("Срок вклада (дней) " );
    scanf("%i", &srok);
    printf("Процентная ставка (годовых) " );
    scanf("%f", &stavka);
    dohod = summ * stavka / 365 / 100 * srok; // 365 - кол-во дней в году
    summ = summ + dohod;
    printf("Доход: %9.2f руб. \n", dohod);
    printf("Сумма по окончании срока вклада: %9.2f руб. \n", summ);
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}
```

2.2. Алгоритмы с разветвляющейся структурой

2.2.1. Инструкция если (*if*)

При разработке алгоритма и написании программы необходимо помнить о том, что:

- инструкция **если** (*if*) используется для выбора одного из двух направлений дальнейшего хода алгоритма или программы;
- выбор последовательности инструкций осуществляется в зависимости от значения условия – заключенного в скобки выражения, записанного после ***if***;
- инструкция, записанная после **иначе** (*else*), выполняется в том случае, если значение выражения условие равно нулю, во всех остальных случаях выполняется инструкция, следующая за условием;
- если при соблюдении или несоблюдении условия надо выполнить не-

сколько инструкций программы, то эти инструкции следует объединить в группу – заключить в фигурные скобки;

- при помощи вложенных одна в другую нескольких инструкций *if-else* можно реализовать множественный выбор.

Пример 4. Составить алгоритм и написать программу решения квадратного уравнения вида $(a \cdot x^2 + b \cdot x + c = 0)$. Программа должна проверять правильность исходных данных и в случае, если коэффициент при второй степени неизвестного равен нулю, выводить соответствующее сообщение.

Схема алгоритма решения квадратного уравнения:

алг Решение квадратного уравнения (**арг вещ** a, b, c ,
вещ рез $x1, x2$)

нач

вещ d

ввод a, b, c

$d = b * b - 4 * a * c$

если $d < 0$ **то** выв "Уравнение не имеет корней"

иначе

если $d = 0$ **то** $x1 = x2 = -b / (2 * a)$

иначе

$x1 = (-b + \sqrt{d}) / (2 * a)$

$x2 = (-b - \sqrt{d}) / (2 * a)$

все

все

вывод $x1, x2$

кон

Программа решения квадратного уравнения:

```
// Решение квадратного уравнения
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    float a, b, c;    // коэффициенты уравнения
    float x1, x2;    // корни уравнения
    float d;         // дискриминант
    printf("\nРешение квадратного уравнения \n ");
    printf("Введите в одной строке значения коэффициентов\n ");
```

```

scanf("%f%f%f", &a, &b, &c); // ввод коэффициентов
d = b*b - 4*a*c; // дискриминант
if (d < 0)
    printf("Уравнение не имеет решения\n ");
else
    if (d==0)
        x1=x2=-b/(2*a);
    else
    {
        x1 = (-b + sqrt(d))/(2*a);
        x2 = (-b - sqrt(d))/(2*a);
    }
printf("Корни уравнения: x1=%3.2f x2=%3.2f \n", x1, x2);
printf("\nДля завершения нажмите на любую клавишу");
while(!kbhit());
}

```

Пример 5. Составить алгоритм и написать программу вычисления частного двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю), выдавать сообщение об ошибке.

Схема алгоритма вычисления частного двух чисел показана на рис. 12:

Программа вычисления частного двух чисел:

```

// Вычисление частного
#include <stdio.h>
#include <conio.h>
void main ()
{
    float a, b, c; // делимое, делитель и частное
    printf("\nВычисление частного\n");
    printf("Введите в одной строке делимое и делитель\n");
    scanf("%f%f", &a, &b);
    if (b != 0)
    {
        c = a / b;
        printf("частное от деления %5.2f на %5.2f ", a, b);
        printf("равно %5.2f", c);
    }
    else
    {
        printf("Ошибка! Делитель не должен быть равен нулю! \n ");
    }
}

```

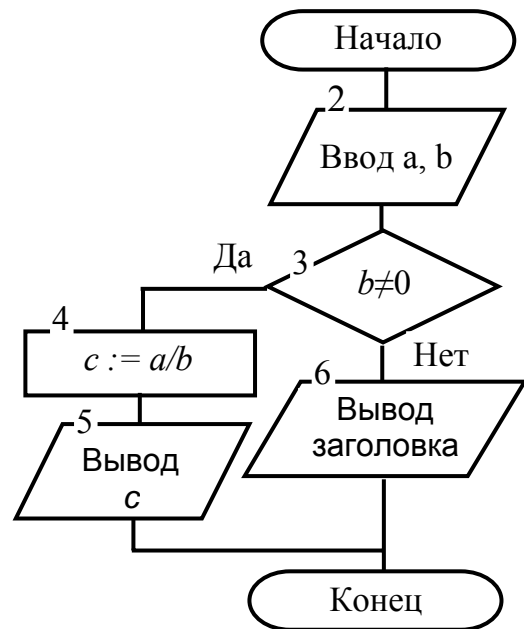


Рис. 12

```

printf("\nДля завершения нажмите на любую клавишу");
while(!kbhit());
}

```

Пример 6. Составить алгоритм и написать программу вычисления стоимости разговора по телефону с учетом 20% скидки, предоставляемой по субботам и воскресеньям.

Схема алгоритма вычисления стоимости разговора по телефону:

алг Вычисление стоимости разговора по телефону (**арг инт** *time*, *day*,
рез вещ *summa*)

нач

ввод *time*, *day*

summa = 2,3 * *time*

если *day* = 6 **или** *day* = 7 **то** *summa* = *summa* * 0,8

все

вывод *summa*

кон

Программа вычисления стоимости разговора по телефону:

```

// Вычисление стоимости телефонного, разговора с учетом
// скидки, предоставляемой по субботам и воскресеньям
#include <stdio.h>
#include <conio.h>
void main()
{
    int time;      // длительность разговора
    int day;       // день недели
    float summa;   // стоимость разговора
    printf("\nВычисление стоимости разговора по телефону\n");
    printf("Введите исходные данные: \n ");
    printf("Длительность разговора (целое кол-во минут) \n ");
    scanf("%i", &time);
    printf("День недели (1-понедельник,...,7-воскресенье) \n ");
    scanf("%i", &day);
    summa = 2.3 * time; // цена минуты 2.3 руб.
    if (day == 6 || day == 7)
    {
        printf("Предоставляется скидка 20%\n ");
        summa = summa * 0.8;
    }
    printf("Стоимость разговора: %3.2f руб. \n", summa);
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}

```

2.2.2. Инструкция выбор (*switch*)

При разработке алгоритма и написании программы необходимо помнить о том, что:

- инструкция выбор (*switch*) предназначена для выбора одного из нескольких возможных направлений дальнейшего хода программы;
- выбор последовательности инструкций осуществляется в зависимости от равенства значения переменной-селектора константе, указанной после слова *case*;
- если значение переменной-селектора не равно ни одной из констант, записанных после *case*, то выполняются инструкции, расположенные после слова *default*;
- в качестве переменной-селектора можно использовать переменную целого или символьного типа.

Пример 7. Составить алгоритм и написать программу, которые запрашивают у пользователя номер дня недели, затем выводят название дня недели или сообщение об ошибке, если введены неверные данные.

Схема алгоритма, который запрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке, если введены неверные данные:

алг Определение дня недели (арг инт *nd*, рез лит "День недели")

нач

ввод *nd*

выбор

при *nd* = 1 вывод "Понедельник"

при *nd* = 2 вывод "Вторник"

при *nd* = 3 вывод "Среда"

при *nd* = 4 вывод "Четверг"

при *nd* = 5 вывод "Пятница"

при *nd* = 6 вывод "Суббота"

при *nd* = 7 вывод "Воскресенье"

иначе вывод "Число должно быть в диапазоне 7"

все

кон

Программа, которая запрашивает у пользователя номер дня недели, затем

выводит название дня недели или сообщение об ошибке, если введены неверные данные:

```
// Выводит название дня недели
#include <stdio.h>
#include <conio.h>
void main()
{
    int nd;          // номер дня недели
    puts("\nВведите номер дня недели ( 1 .. 7 ) ");
    scanf("%i", &nd);
    switch (nd)
    {
        case 1:      puts("Понедельник \n");      break;
        case 2:      puts("Вторник \n");           break;
        case 3:      puts("Среда \n");             break;
        case 4:      puts("Четверг \n");           break;
        case 5:      puts("Пятница \n");           break;
        case 6:      puts("Суббота \n");           break;
        case 7:      puts("Воскресенье \n");       break;
        default:     puts("Число должно быть в диапазоне .. 7 \n ");
    }
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}
```

Пример 8. Составить алгоритм и написать программу, которые вычисляют стоимость междугородного телефонного разговора (цена одной минуты определяется расстоянием до города, в котором находится абонент). Исходными данными для программы являются код города и длительность разговора.

Схема алгоритма вычисления стоимости междугородного телефонного разговора показана на рис. 13.

Программа вычисления стоимости междугородного телефонного разговора:

```
// Определение стоимости междугородного телефонного разговора
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main()
{
    int kod;          // код города
    float cena;       // цена минуты
    int dlit;         // длительность разговора
    float summ;       // стоимость разговора
    printf("\nВычисление стоимости разговора по телефону\n");
    printf("Введите исходные данные : \n");
    printf("Длительность разговора (целое кол-во минут)",
```

```

scanf("%i", &dilit);
cout << "Код города" << "\n ";
cout << "Москва" << "095" << "\n ";
cout << "Саратов" << "8452" << "\n";
cout << "Омск" << "3812" << "\n ";
cout << "Иркутск" << "3952" << "\n ";
scanf("%i", &kod);

```

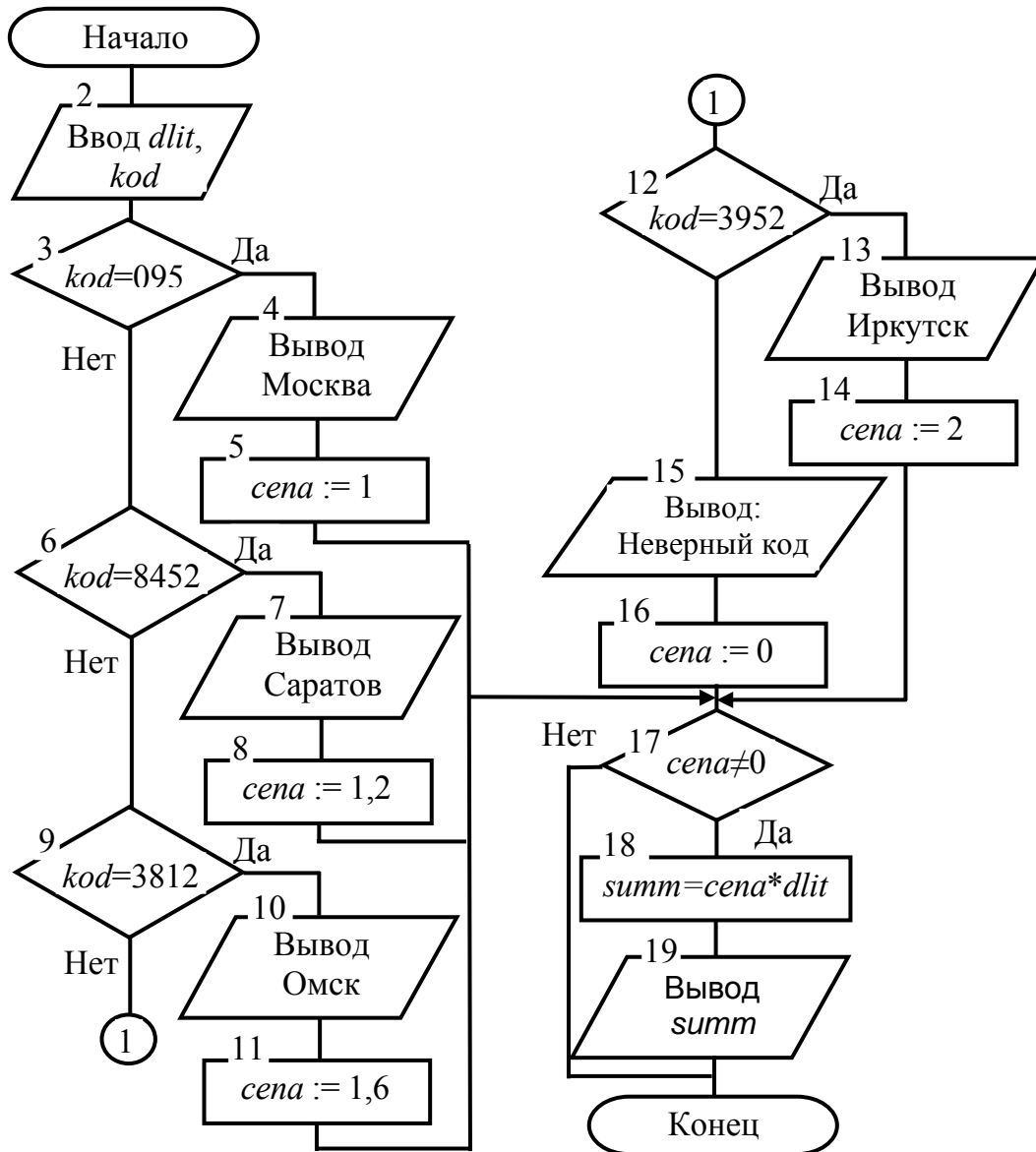


Рис. 13

```

switch (kod)
{
    case 095:
        cout << " Москва";
        cena = 1;
        break;

    case 8452:
        cout << " Саратов ";
        cena = 1.2;
        break;

    case 3812:
        cout << " Омск ";

```

```

                                cena = 1.6;
                                break;
case 3952:                      cout << " Иркутск ";
                                cena = 2.0;
                                break;
default:                        printf("неверно введен код.");
                                cena = 0 ;
                                }
if (cena != 0)
{
    summ = cena * dlit;
    printf("Цена минуты: %i руб. \n", cena);
    printf("Стоимость разговора: %3.2f руб. \n", summ);
}
printf("\nДля завершения нажмите на любую клавишу");
while(!kbhit());
}

```

Пример 9. Составить алгоритм и написать программу, которые по дате определяют день недели, на который эта дата приходится. Для вычисления дня недели воспользуйтесь формулой:

$$(d + \left\lceil \frac{1}{5}(13 \cdot m - 1) \right\rceil + Y + \left\lceil \frac{Y}{4} \right\rceil + \left\lceil \frac{c}{4} \right\rceil - 2 \cdot c + 777) \cdot \text{mod } 7.$$

Здесь d – число месяцев, m – номер месяца, если начинать счет с марта, как это делали в Древнем Риме (март – 1, апрель – 2, ..., февраль – 12), Y – номер года в столетии, c – количество столетий. Квадратные скобки означают, что надо взять целую часть от значения, находящегося в скобках. Вычисленное по формуле значение определяет день недели: 1 – понедельник, 2 – вторник, ..., 6 – суббота, 0 – воскресенье.

Схема алгоритма, который по дате определяет день недели, на который эта дата приходится:

алг День недели (**арг** **int** $day, month, year$, **рез** **лит** "День недели")

нач

int c, y, m, d

ввод $day, month, year$

если $month = 1$ **или** $month = 2$ **то** $year = year - 1$

все

$m = month - 2$

если $m \leq 0$ **то** $m = m + 12$

все

$c = year/100$

$y = year - c*100$

$d = (day + (13*m - 1)/5 + y + y/4 + c/4 - 2*c + 777)\%7$

выбор

при $d = 1$ **вывод** "Понедельник"

при $d = 2$ **вывод** "Вторник"

при $d = 3$ **вывод** "Среда"

при $d = 4$ **вывод** "Четверг"

при $d = 5$ **вывод** "Пятница"

при $d = 6$ **вывод** "Суббота"

при $d = 0$ **вывод** "Воскресенье"

все

кон

Программа, которая по дате определяет день недели, на который эта дата приходится:

```
// По дате определяет день недели
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main()
{
    int day, month, year;    // день, месяц, год
    int c, y;                // столетие и год в столетии
    int m;                  // месяц по древнеримскому календарю
    int d;                  // день недели
    cout << "Определение дня недели по дате" << "\n";
    cout << "Введите дату: день месяц год" << "\n";
    cout << "Например, 12 1 2003" << "\n";
    scanf("%i%i%i", &day, &month, &year);
    if (month == 1 || month == 2)
        year--;              // январь и февраль относятся
                             // к предыдущему году
    m = month - 2;           // год начинается с марта
    if (m <= 0) m += 12;     // для января и февраля
                             // здесь m - номер месяца по римскому календарю

    c = year / 100;
    y = year - c*100;
    d = (day + (13*m - 1)/5 + y + y/4 + c/4 - 2*c + 777)%7;
    switch (d)
    {
        case 1:    cout << "Понедельник" << "\n"; break;
        case 2:    cout << "Вторник" << "\n";      break;
```

```

        case 3:    cout << "Среда" << "\n";        break;
        case 4:    cout << "Четверг" << "\n";        break;
        case 5:    cout << "Пятница" << "\n";        break;
        case 6:    cout << "Суббота" << "\n";        break;
        case 0:    cout << "Воскресенье" << "\n";
    }
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}

```

2. 3. Задачи для самостоятельного решения

1. Вычислите длину окружности, площадь круга и объем шара одного и того же заданного радиуса.
2. Вычислите периметр и площадь прямоугольного треугольника по двум катетам.
3. По координатам трех вершин некоторого треугольника найдите его площадь и периметр.
4. Вычислите дробную часть среднего геометрического трех заданных вещественных чисел.
5. Определите, является ли заданное целое число A нечетным двузначным числом.
6. Определите, имеется ли среди заданных целых чисел A , B , C хотя бы одно четное.
7. Даны три числа. Выберите те из них, которые принадлежат заданному отрезку $[a, b]$.
8. Определите число, полученное выписыванием в обратном порядке цифр заданного целого трехзначного числа.
9. Вычислите площадь кольца, ширина которого равна h , а отношение радиуса большей окружности к радиусу меньшей окружности равно d .
10. Определите, есть ли среди цифр заданного целого трехзначного числа одинаковые.
11. Заданы площади круга и квадрата. Определите, поместится ли квадрат в круге.
12. Заданы координаты двух точек. Определите, лежат ли они на одной окружности с центром в начале координат.
13. Определите, лежит ли данная точка на одной из сторон треугольника, заданного координатами его вершин.
14. Выберите наибольшее из трех заданных чисел.

15. Значения заданных переменных a , b и c перераспределите таким образом, что a , b , c станут соответственно наименьшим, средним и наибольшим значениями.

16. Решите биквадратное уравнение $ax^4 + bx^2 + c = 0$.

17. Определите, пройдет ли кирпич с ребрами a , b , c в прямоугольное отверстие со сторонами x и y .

18. Идет k -я секунда суток. Определите, сколько полных часов и полных минут прошло к этому моменту.

2.4. Алгоритмы, реализуемые с помощью циклов

2.4.1. Инструкция для (*for*)

При разработке алгоритма и написании программы необходимо помнить о том, что:

- инструкция `for` используется для организации циклов с фиксированным, известным во время разработки алгоритма (программы), числом повторений;
- количество повторений цикла определяется начальным значением переменной-счетчика и условием завершения цикла;
- переменная-счетчик должна быть целого типа и может быть объявлена непосредственно в инструкции цикла.

Пример 10. Составить алгоритм и написать программу, которые вычисляют сумму первых n целых положительных целых чисел. Количество суммируемых чисел должно вводиться во время работы программы.

Схема алгоритма вычисления суммы первых n целых положительных целых чисел:

алг Вычисление суммы первых n целых положительных чисел (арг цел n ,
рез цел $summ$)

нач

цел i

ввод n

нц для i от 1 до n

$summ = summ + i$

кц

вывод $summ$

кон

Программа вычисления суммы первых n целых положительных целых

чисел:

```
// Вычисляет сумму первых n целых положительных чисел
#include <stdio.h>
#include <conio.h>
void main()
{
    int n;           // количество суммируемых чисел
    int summ;        // сумма
    int i;           // счетчик циклов
    printf("Вычисление суммы положительных чисел\n");
    printf("Введите количество суммируемых чисел \n");
    scanf("%i", &n);
    summ = 0;
    for (i = 1; i <= n; i++)
        summ = summ + i;
    printf("Сумма первых %i целых положительных чисел ", n);
    printf("равна %i", summ);
    printf("\nДля завершения нажмите на любую клавишу");
    while(!kbhit());
}
```

Пример 11. Составить алгоритм и написать программу, которые выводят таблицу значений функции $y = x^2$ в диапазоне от -10 до 10 , с шагом $0,5$.

Схема алгоритма вычисления и вывода таблицы значений функции $y = x^2$ в диапазоне от -10 до 10 , с шагом $0,5$ показана на рис. 14.

Программа вычисления и вывода таблицы значений функции $y = x^2$:

```
// Вывод таблицы функции  $y = x^2$ 
#include <stdio.h>
#include <conio.h>
#define LB -10.0 // нижняя граница диапазона изменения // аргумента
#define HB 10.0 // верхняя граница диапазона изменения // аргумента
#define DX 0.5 // приращение аргумента
void main()
{
    float x,y; // аргумент и значение функции
    int n; // кол-во точек
    int i; // счетчик циклов
    n = (HB - LB)/DX + 1;
    x = LB;
    for (i = 1; i <= n; i++)
```

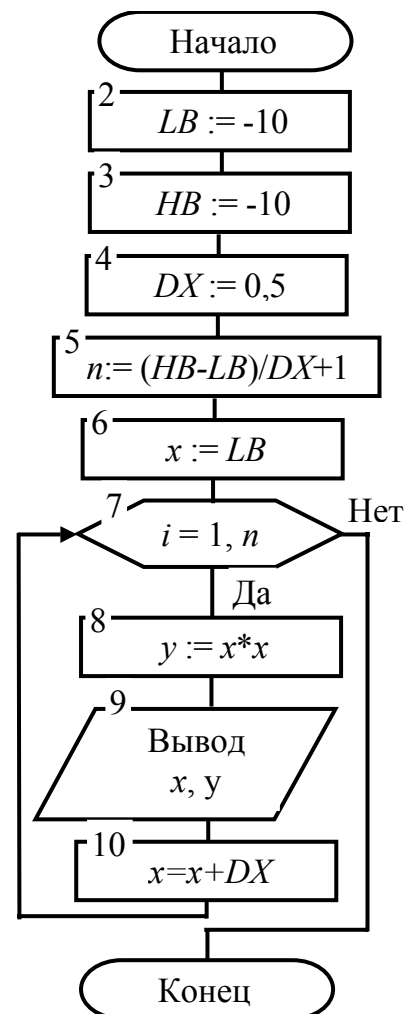


Рис. 14

```

{
    y = x*x;
    printf("x=%6.2f  y=%6.2f\n", x, y);
    x += DX;
}
printf("\nДля завершения нажмите на любую клавишу");
getch();
}

```

Пример 12. Составить алгоритм и написать программу, которые вычисляют среднее арифметическое последовательности чисел, вводимых с клавиатуры. После ввода последнего числа программа должна вывести минимальное и максимальное число последовательности. Количество чисел последовательности должно задаваться во время работы программы.

Схема алгоритма вычисления среднего арифметического последовательности чисел, вводимых с клавиатуры и определения минимального и максимального числа последовательности:

алг Вычисление среднего арифметического последовательности чисел (**арг цел** n , **арг вещ** a , **рез вещ** $sred$, min , max)

нач

вещ sum , **цел** i

ввод n , a

$min = a$

$max = a$

$sum = a$

нц для i **от** 1 **до** n

$sum = sum + a$

если $a < min$ **то** $min = a$ **все**

если $a > max$ **то** $max = a$ **все**

кц

$sred = sum/n$

вывод n , $sred$, min , max

кон

Программа вычисления среднего арифметического последовательности чисел, вводимых с клавиатуры и определения минимального и максимального числа последовательности:

```

// Вычисляет среднее арифметическое и определяет
// минимальное и максимальное число последовательности

```

```

// дробных чисел, вводимых с клавиатуры
#include <stdio.h>
#include <conio.h>
void main ()
{
    float a;                // очередное число
    int n;                  // количество чисел
    float sum;              // сумма введенных чисел
    float sred;             // среднее арифметическое
    float min;              // минимальное число последовательности
    float max;              // максимальное число последовательности
    int i;                  // счетчик циклов
    printf("Обработка последовательности дробных чисел \n");
    printf("Введите количество чисел последовательности - "),
    scanf("%i", &n);
    printf("Введите первое число последовательности чисел \n");
    scanf("%f", &a);        // вводим первое число последовательности
                            // предположим, что:
    min = a;                // пусть первое число является минимальным
    max = a;                // пусть первое число является максимальным
    sum=a;
    printf("Введите первое число последовательности чисел \n");
    for (i = 1; i < n; i++)
    {
        scanf("%f", &a);
        sum += a;
        if (a < min) min = a;
        if (a > max) max = a;
    }
    sred = sum/n;
    printf("Количество чисел: %i\n", n);
    printf("Среднее арифметическое = %6.2f\n", sred);
    printf("Минимальное число = %6.2f\n", min);
    printf ("Максимальное число = %6.2f\n", max);
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}

```

2.4.2. Инструкция выполнять пока (*do-while*)

При разработке алгоритма и написании программы необходимо помнить о том, что:

- число повторений инструкций цикла *do-while* определяется ходом выполнения программы;
- инструкция цикла *do-while* выполняется до тех пор, пока значение выражения, записанного после оператора *while*, не станет равным нулю;
- после оператора *while* надо записывать условие выполнения инструкций

цикла;

- для завершения цикла **do-while** в теле цикла обязательно должны быть инструкции, выполнение которых влияет на условие завершения цикла;

- цикл **do-while** – это цикл с постусловием, т. е. инструкции тела цикла будут выполнены хотя бы один раз;

- цикл **do-while**, как правило, используется для организации приближенных вычислений, в задачах поиска и обработки данных, вводимых с клавиатуры или из файла.

Пример 13. Составить алгоритм и написать программу, которые вычисляют сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры.

Схема алгоритма вычисления суммы и среднего арифметического последовательности положительных чисел, вводимых с клавиатуры:

алг Вычисление суммы и среднего арифметического последовательности положительных чисел (**арг цел** a , **рез цел** n , sum , **рез вещ** m)

нач

вещ s

$s = 0$

$n = 0$

нц пока $a > 0$

ввод a

если $a > 0$ **то**

$s = s + a$

$n = n + 1$

все

кц

вывод n, s

$m = s/n$

вывод m

кон

Программа вычисления суммы и среднего арифметического последовательности положительных чисел, вводимых с клавиатуры:

```
// Вычисление среднего арифметического
// последовательности положительных чисел
#include <stdio.h>
```

```

#include <conio.h>
void main()
{
    int a;                // число, введенное с клавиатуры
    int n;                // количество чисел
    int s;                // сумма чисел
    float m;              // среднее арифметическое
    s = 0;
    n = 0;
    printf("\nВычисление среднего арифметического");
    printf("последовательности положительных чисел \n");
    printf("Вводите числа. Для завершения введите ноль \n");
    do {
        scanf("%i", &a);
        if (a > 0)
        {
            s += a;
            n++;
        }
    } while (a > 0);
    printf("Введено чисел = %i\n", n);
    printf("Сумма чисел= %i\n", s);
    m = (float)s/n;
    printf("Среднее арифметическое= %6.2f", m);
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}

```

Пример 14. Составить алгоритм и написать программу, которые определяют максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности неограниченна).

Схема алгоритма определения максимального числа из введенной с клавиатуры последовательности положительных чисел показана на рис. 15.

Программа определения максимального числа из введенной с клавиатуры последовательности положительных чисел:

```

// Определение максимального числа
// в последовательности положительных чисел
#include <stdio.h>
#include <conio.h>

```

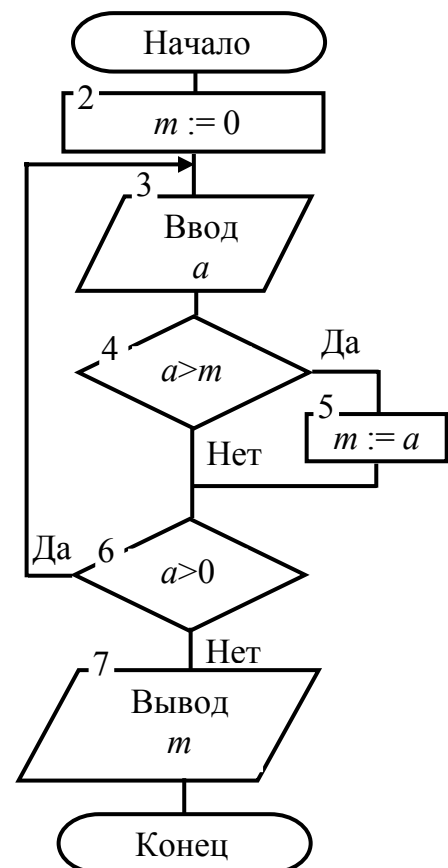


Рис. 15


```

void main()
{
    int a;                // очередное число
    int m;                // максимальное число
    puts("\nОпределение максимального числа");
    puts("последовательности положительных чисел.");
    puts("Вводите числа. Для завершения введите ноль.");
    m = 0;
    do {
        scanf("%i", &a);
        if (a > m) m = a;
    } while (a > 0);
    printf("Максимальное число: %i", m);
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}

```

Пример 15. Составить алгоритм и написать программу, которые проверяют, является ли введенное пользователем целое число простым.

Схема алгоритма который проверяет, является ли введенное пользователем целое число простым:

алг Проверка, является ли число простым (**арг цел** n , **рез цел** n)

нач

цел d, r

ввод n

$d = 2$

нц пока $r \neq 0$

ввод a

$r = n \% d$

если $r \neq 0$ **то** $d = d + 1$

все

кц

если $d = n$ **то вывод** " n – простое число"

иначе вывод " n – не простое число"

все

кон

Программа, которая проверяет, является ли введенное пользователем целое число простым:

```

// Проверяет, является ли число простым
#include <stdio.h>
#include <conio.h>

```

```

void main()
{
    int n;           // число
    int d;           // делитель
    int r;           // остаток от деления n на d
    printf("Введите целое число - ");
    scanf("%i", &n);
    d = 2;           // сначала будем делить на два
    do {
        r = n % d;
        if (r != 0) d++;
    } while ( r != 0 ); // пока n не разделится на d
    if (d == n)
        printf("%i - простое число", n);
    else
        printf("%i - не простое число", n);
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}

```

2.4.3. Инструкция пока (*while*)

При разработке алгоритма и написании программы необходимо помнить о том, что:

- инструкция цикла ***while*** выполняется до тех пор, пока значение выражения, записанного после оператора ***while***, не станет равным нулю;
- после оператора ***while*** надо записывать условие выполнения инструкций цикла;
- для завершения цикла ***while*** в теле цикла обязательно должны быть инструкции, выполнение которых влияет на условие завершения цикла;
- цикл ***while*** — это цикл с постусловием, т. е. возможна ситуация, при которой инструкции тела цикла ни разу не будут выполнены;
- цикл ***while***, как правило, используется для организации приближенных вычислений, в задачах поиска и обработки данных, вводимых с клавиатуры или из файла.

Пример 16. Составить алгоритм и написать программу, которые выводят на экран таблицу значений функции $y = x^3$ в диапазоне от -8 до 8 . Шаг изменения аргумента $0,25$.

Схема алгоритма вычисления и вывода таблицы значений функции $y = x^3$:

алг Вычисление значений функции $y = x^3$ (**арг вещ** $x1, x2, dx$, **рез вещ** y)

нач

вещ x

$x1 = -8$

$x2 = 8$

$dx = 0,25$

$x = x1$

нц пока $x > x2$

$y = x * x * x$

ВЫВОД x, y

$x = x + dx$

кц

кон

Программа вычисления и вывода таблицы значений функции $y = x^3$:

```
// Выводит таблицу функции  $y = x^3$ 
#include <stdio.h>
#include <conio.h>
void main()
{
    float x, dx;           // аргумент и его приращение
    float x1, x2;          // диапазон изменения аргумента
    float y;               // значение функции
    x1 = -8;
    x2 = 8;
    dx = 0.25;
    x = x1;
    while(x < x2)
    {
        y = x * x * x;
        printf("x=%3.2f  y=%3.2f\n", x, y);
        x += dx;
    }
    printf("\nДля завершения нажмите на любую клавишу");
    getch();
}
```

Пример 17. Составить алгоритм и написать программу, которые вычисляют число π с заданной пользователем точностью ($\varepsilon = 0,001$). Для вычисления значения числа π воспользуйтесь тем, что значение частичной суммы ряда:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

при суммировании достаточно большого количества членов приближается к значению $\pi/4$.

Схема алгоритма вычисления числа π с заданной пользователем точно-

стью ($\varepsilon = 0,001$) показана на рис. 16.

```
// Вычисление числа "Пи"
#include <stdio.h>
#include <conio.h>
```

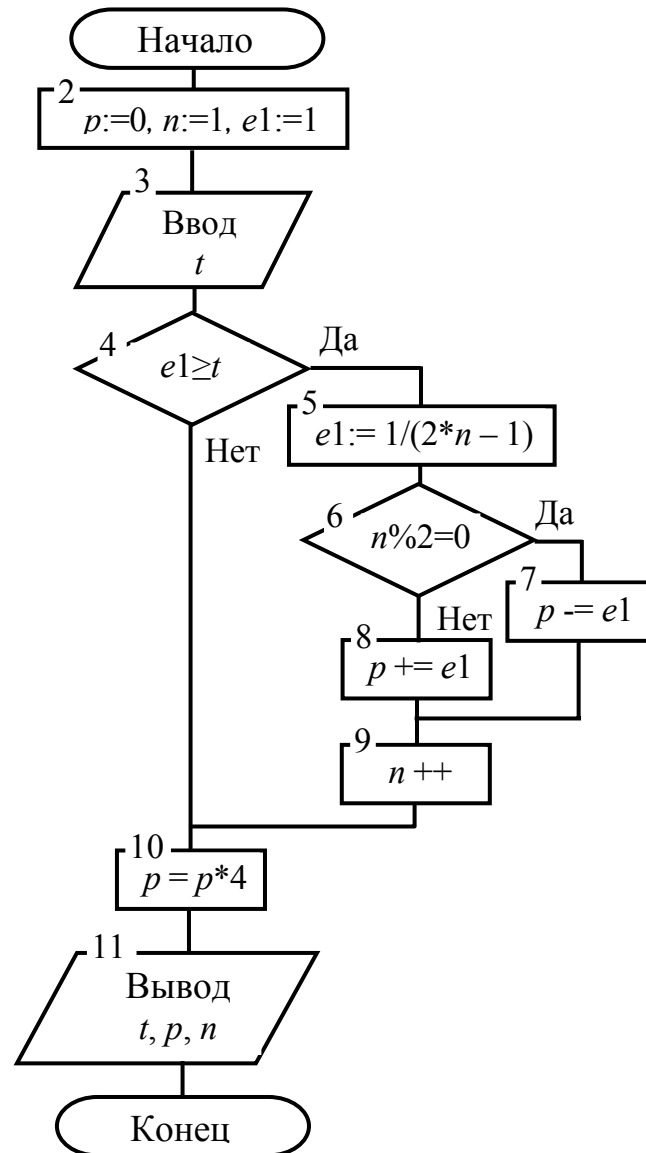


Рис. 16

```
void main ()
{
    float p;           // вычисляемое значение ПИ
    float t;           // точность вычисления
    int n;              // номер члена ряда
    float e1;          // значение члена ряда
    p = 0;
    n = 1;
    e1 = 1;             // начальное значение
    printf("\nЗадайте точность вычисления ПИ – ")
    scanf("%f", &t);
    printf("Вычисление ПИ с точностью %f\n",t);
    while (e1 >= t )
    {
```

```

e1 = (float) 1 / (2*n -1);
if ((n % 2) == 0)
    p -= e1;
else p += e1;
n++;
}
p = p*4;
printf("\nЗначение ПИ с точностью %f равно %f\n", t, p);
printf("Просуммировано %i членов ряда.\n", n);
printf("\nДля завершения нажмите на любую клавишу");
while(!kbhit());
}

```

Пример 18. Составить алгоритм и написать программу, которые вычисляют наибольший общий делитель двух целых чисел.

Схема алгоритма вычисления наибольшего общего делителя двух целых чисел:

алг Наибольший общий делитель двух целых чисел (**арг цел** $n1, n2$,
рез цел nod)

нач

цел r

нц пока $n1 \% n2$

$r = n1 \% n2$

$n1 = n2$

$n2 = r$

кц

$nod = n2$

вывод nod

кон

Программа вычисления наибольшего общего делителя двух целых чисел:

```

// Вычисление наибольшего общего делителя
// двух целых чисел (алгоритм Евклида)
#include <stdio.h>
#include <conio.h>
void main ()
{
    int n1, n2;           // числа, НОД которых надо вычислить
    int nod;              // наибольший общий делитель
    int r;                // остаток от деления n1 на n2
    printf("\nВычисление наибольшего общего делителя " );
    printf("для двух целых чисел \n");
    printf("Введите в одной строке два числа ");
}

```

```

scanf("%i%i", &n1, &n2);
printf("НОД чисел %i и %i - это ", n1, n2);
while (n1 % n2)
{
    r = n1 % n2;          // остаток от деления
    n1 = n2;
    n2 = r;
}
nod = n2;
printf("%i\n", nod);
printf("\nДля завершения нажмите на любую клавишу");
while(!kbhit());
}

```

2. 5. Задачи для самостоятельного решения

1. Подсчитайте число и сумму положительных, число и произведение отрицательных элементов заданного массива $A(N)$.

2. Элементы заданного массива $B(N)$ перепишите в новый массив $A(N)$ в обратном порядке.

3. Из заданного вектора $A(3N)$ получите вектор $B(N)$, очередная компонента которого равна среднему арифметическому очередной тройки компонент вектора A .

4. В заданном массиве $X(N)$ замените нулями все отрицательные компоненты, непосредственно предшествующие его максимальной компоненте (первой по порядку, если их несколько).

5. Вычислите сумму квадратов всех элементов заданного массива $X(N)$, за исключением элементов, кратных пяти.

6. Вычислите значения функции $z = (a+b+c)/i$, если a изменяется от 0 с шагом 0,1; b изменяется от 5 с шагом 0,5; c изменяется от 10 с шагом 1. При этом a , b и c изменяются одновременно i .

7. В заданном массиве $A(N)$ поменяйте местами наибольший и наименьший элементы (первые по порядку, если их несколько).

8. В заданном массиве $A(N)$ определите количество элементов, которые меньше заданного значения.

9. Дан массив $A(N)$. Получите массив $B(N)$, i -й элемент которого равен среднему арифметическому первых i элементов массива A : $b = (a_1 + a_2 + \dots + a_i)/i$

10. Вычислите значения полиномов $P = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_1 \cdot x + a_0$

$Q = a_0 \cdot x^n + a_1 \cdot x^{n-1} + \dots + a_{n-1} \cdot x + a_n$, используя формулу Горнера. Коэффициенты полинома заданы в виде вектора $A = (a_0, a_1, \dots, a_n)$.

11. Запишите подряд в массив $A(N)$ элементы заданного массива $B(2N)$, стоящие на четных местах, а элементы, стоящие на нечетных местах, запишите в массив $C(N)$.

12. В заданном массиве $A(N)$ вместо a_1 запишите наибольший элемент массива, а вместо a_N – наименьший элемент массива.

13. В заданном массиве $A(N)$, все элементы которого попарно различны, найдите: а) наибольший элемент из отрицательных; б) наименьший элемент из положительных; в) второй по величине элемент.

14. В заданном массиве $A(N)$ определите число соседств: а) двух положительных чисел; б) двух чисел разного знака; в) двух чисел одного знака, причем абсолютная величина первого числа должна быть больше второго числа; г) четного числа и нечетного с нечетным индексом.

15. В заданном массиве $A(N)$ положительные элементы уменьшите вдвое, а отрицательные элементы замените на значения их индексов.

16. В заданном массиве $A(N)$ вычислите среднее геометрическое и среднее арифметическое значения для положительных элементов.

17. Вычислите $P = 1 \cdot 2 + 2 \cdot 3 \cdot 4 + 3 \cdot 4 \cdot 5 \cdot 6 + \dots + N \cdot (N + 1) \cdot \dots \cdot 2N$.

18. Дан вектор $A(N)$. Найдите порядковый номер того из элементов, который наиболее близок к какому-нибудь целому числу (первого по порядку, если таких несколько).

19. Для заданного набора коэффициентов a, b, c, d найдите наименьшее значение функции $y = ax^3 + bx^2 + cx + d$ и значение аргумента, при котором оно получено. Значение x изменяется от 0 до 2 с шагом 0,2.

20. У кассы Аэрофлота выстроилась очередь из N человек. Время обслуживания кассиром i -го клиента равно T_i ($i = 1, \dots, N$). а) Определите время пребывания в очереди каждого клиента, б) Укажите номер клиента, для обслуживания которого кассиру потребовалось больше всего времени.

21. В соревнованиях по фигурному катанию N судей независимо выставляют оценки спортсмену. Затем из объявленных оценок удаляют самую высокую (одну, если самую высокую оценку выставили несколько судей). Аналогично поступают с самой низкой оценкой. Для оставшихся оценок вычисляется среднее арифметическое, являющееся зачетной оценкой. По за-

данным оценкам судей определите зачетную оценку спортсмена.

22. Дана матрица $A(N,M)$. Найдите ее наибольший элемент (первый по порядку, если их несколько) и номера строки и столбца, на пересечении которых он находится.

23. В каждой строке заданной матрицы $A(N,M)$ вычислите сумму, количество и среднее арифметическое положительных элементов.

24. Для заданной целочисленной матрицы $A(N,M)$ определите, является ли сумма ее элементов четным числом, и выведите на печать соответствующий текст.

25. Дана матрица $A(N,M)$. Найдите количество элементов этой матрицы, больших среднего арифметического всех ее элементов.

26. Дана целочисленная матрица $A(N,M)$. Вычислите сумму и произведение тех ее элементов, которые при делении на два дают нечетное число.

27. В заданной матрице $A(N,M)$ поменяйте местами столбцы с номерами P и Q .

28. Дана матрица $A(N,M)$. Вычислите вектор $X(M)$, где значение X_j равно сумме положительных элементов j -го столбца матрицы A .

29. Дана матрица $A(N,M)$. Получите вектор $X(M)$, равный P -й строке матрицы, и вектор $Y(N)$, равный Q -му столбцу матрицы.

30. Дана матрица $A(N,M)$. Поменяйте местами ее наибольший и наименьший элементы (первые по порядку, если их несколько). Порядок просмотра элементов матрицы слева направо и сверху вниз.

31. Заданной матрицы $A(N,N)$ найдите: а) сумму всех элементов; б) сумму элементов главной диагонали; в) значения наибольшего и наименьшего из элементов главной диагонали.

32. Заданной матрицы $A(N,N)$ найдите сумму элементов, расположенных в строках с отрицательным элементом на главной диагонали.

33. Дана матрица $A(N,M)$. Определите: а) число ненулевых элементов в каждой строке матрицы; б) общее число ненулевых элементов в матрице; в) отношение числа ненулевых элементов в каждой строке матрицы к общему числу ненулевых элементов в матрице.

34. Вычислите матрицу $A(N,M)$, являющуюся суммой матриц $A(N,N)$ и $B(N,N)$.

35. В заданном массиве $X(N,M)$ все числа различны. В каждой строке

выбирается минимальный элемент, затем среди этих чисел выбирается максимальное. Напечатайте номер строки массива X , в которой расположено выбранное число.

36. В заданном массиве $A(N, M)$ переставьте строки так, чтобы суммы их элементов возрастали.

37. В заданном массиве $A(N, N)$ вычислите две суммы элементов, расположенных выше и ниже побочной диагонали.

38. Задан список участников соревнований по плаванию и их результаты. Все результаты различны. Расположите результаты и фамилии участников в соответствии с занятым местом.

39. На основе сведений о ежедневном пробеге на тренировке спортсменов команды рассчитайте среднесуточный и общий пробег каждого спортсмена за 20 дней.

40. Известен расход по N видам горючего в каждом из M автохозяйств. Определите для каждого хозяйства вид горючего с наибольшим и с наименьшим расходом.

41. На основе сведений об отметках учеников класса в последней четверти: а) вычислите средние баллы по каждому предмету; б) определите группу из трех лучших учеников; в) определите группу из трех самых слабых учеников.

42. Шестизначный номер автобусного билета называют счастливым, если равны суммы его первых трех и последних трех цифр. Подсчитайте количество счастливых билетов.

43. Вычислите сумму $Z = 1 + 2 + 3 + \dots$. Вычисления прекратите, когда значение Z превысит заданное значение A .

44. Проверьте, есть ли в заданной целочисленной последовательности a_1, a_2, \dots, a_N элементы, равные нулю. Если есть, найдите номер первого из них, если нет, выдайте соответствующий текст.

45. Выясните, имеются ли в заданном векторе $A(N)$ два подряд идущих нулевых элемента.

46. Выясните, имеются ли в заданном целочисленном векторе $A(N)$ три подряд идущих элемента одного знака.

47. Если у заданного вектора $A(N)$ есть хотя бы один элемент, меньший чем -5 , то все отрицательные элементы замените их квадратами, оста-

вив остальные элементы без изменения; в противном случае вектор домножьте на 0,1.

48. Имеется последовательность чисел a_1, a_2, \dots, a_N . Найдите сумму первых из них (считая слева направо), произведение которых не превышает заданного числа M .

49. Все элементы заданного вектора $A(N)$, начиная с первого по порядку положительного элемента, уменьшите на единицу.

50. Определите, имеются ли среди элементов главной диагонали заданной целочисленной матрицы $A(N,N)$ числа, равные нулю.

51. Найдите любое трехзначное число, кратное заданному P и не равное ему.

52. Если в заданном целочисленном векторе $A(N)$ есть элементы со значением, равным заданному числу B , то переменной C присвойте значение, равное сумме всех элементов, предшествующих первому по порядку такому элементу. В противном случае выведите соответствующий текст.

53. Определите, имеется ли в заданном массиве $A(N)$ хотя бы одна пара соседних чисел, являющихся взаимно-обратными.

54. Определите, имеется ли в заданном целочисленном массиве $X(N)$ число, кратное заданным числам A и B и не кратное числу C .

55. Дано натуральное число N . Выясните, сколько цифр оно содержит.

56. Найдите сумму цифр заданного натурального числа.

57. Проверьте, все ли элементы заданного массива $A(N)$ положительны.

58. Определите по данным музейного каталога, есть ли в музее хотя бы одна картина Левитана или Шишкина. Если есть, выдайте ее название, в противном случае выдайте соответствующий текст.

59. Определите по прейскуранту, можно ли подобрать в спортивном магазине велосипед, стоимость которого не превышает имеющуюся у покупателя сумму.

60. Известен начальный вклад клиента в банк и процент годового дохода. Определите, через сколько лет вклад превысит заданный размер и каков при этом будет размер вклада.

61. Торговая фирма в первый день работы реализовала товаров на p

руб., а затем ежедневно увеличивала выручку на 3%. Какой будет выручка фирмы в тот день, когда она впервые превысит заданное значение q ($q > p$)? Сколько дней придется торговать фирме для достижения этого результата?

62. Малое предприятие в первый день работы выпустило p единиц товарной продукции. Каждый последующий день оно выпускало продукции на q единиц больше, чем в предыдущий. Сколько дней потребуется предприятию, чтобы общее количество выпущенной продукции за все время работы впервые превысило запланированный объем?

63. В заданной целочисленной матрице $A(N, M)$ выведите на печать индексы первого положительного элемента, кратного заданному числу K . Если таких элементов в матрице нет, то выведите соответствующий текст.

64. В заданной целочисленной матрице $A(N, M)$ замените первый отрицательный элемент максимальным элементом матрицы. Если отрицательных элементов нет, то выведите соответствующий текст.

65. Из заданной матрицы $A(N, M)$ удалите строку, в которой находится первый отрицательный элемент.

66. В заданной матрице $A(N, M)$ найдите индексы первого элемента, превосходящего среднее арифметическое всех элементов.

67. Из заданной матрицы $A(N, M)$ удалите строку и столбец, в которых находится первый элемент, равный нулю. Полученную матрицу уплотните.

68. Если в заданной матрице $A(N, M)$ есть хотя бы один элемент, больший ста, то элементы обеих диагоналей замените нулями.

69. Дана целочисленная матрица $A(N, M)$. Найдите номер первой из ее строк, которые начинаются с K положительных чисел подряд.

70. Элементы заданной матрицы $A(N, M)$ переписывайте построчно в одномерный массив до тех пор, пока не встретится нулевой элемент.

71. Заданное натуральное число M представьте в виде суммы квадратов двух неравных натуральных чисел. В случае если это невозможно, выведите соответствующее сообщение.

72. Дана целочисленная матрица $A(N, N)$. Просматривая ее элементы в заданном порядке, найдите первый четный элемент и поменяйте его местами с диагональным элементом той строки, в которой он находится. Порядок просмотра: а) сверху вниз и справа налево; б) снизу вверх и слева направо; в) справа налево и снизу вверх.

73. Проверьте, удовлетворяет ли заданная матрица $A(N,N)$ следующему условию: для всех $i > 1$ и для всех $j > 1$ верно неравенство $a_{ij} > a_{i-1,j} + a_{i,j-1}$.

74. Для заданной матрицы $A(N,N)$ найдите хотя бы одно k , такое, что k -я строка матрицы совпадает с k -м столбцом.

75. Даны три целочисленных массива $A(N)$, $B(M)$ и $C(L)$. Найдите хотя бы одно число, встречающееся во всех трех массивах. Если таких чисел нет, выведите соответствующее сообщение.

76. В школе есть три параллельных десятых класса. Даны списки десяти классников, содержащие фамилию и имя каждого ученика. Выясните: а) в каких классах есть однофамильцы; б) в каких классах есть тезки; в) есть ли в параллельных десятых классах однофамильцы; г) в каких классах есть ученики, у которых совпадают и имя и фамилия; д) есть ли в десятых классах однофамильцы первого космонавта.

77. Дана матрица $A(N,N)$. Переменной B присвойте значение, равное количеству строк матрицы A , содержащих хотя бы одну нулевую компоненту.

78. Дана матрица $B(N,N)$. Получите вектор $A(N)$, компоненты которого находятся по правилу: A_i равно первому по порядку положительному элементу в i -й строке матрицы (если таких элементов в строке нет, то примите $A_i = -1$).

79. Дана матрица $B(N,N)$. Получите вектор $A(N)$, компоненты которого находятся по правилу: A_i равно количеству отрицательных чисел, с которых начинается i -я строка.

80. Среди строк заданной целочисленной матрицы, содержащих только нечетные элементы, найдите строку с максимальной суммой модулей элементов.

81. Среди столбцов заданной целочисленной матрицы, содержащих только такие элементы, которые по модулю не больше 10, найдите столбец с минимальным произведением элементов.

82. В заданной матрице $A(N,M)$ найдите количество строк, не содержащих отрицательных чисел.

83. Дана целочисленная матрица $A(N,N)$. Сформируйте результирующий вектор B , элементами которого являются суммы элементов только тех строк матрицы A , которые начинаются с K положительных чисел подряд.

84. Подсчитайте количество столбцов заданной целочисленной матрицы

$A(N,N)$, в которых имеются взаимно-противоположные соседние числа.

85. Дана матрица $A(N,M)$. Постройте вектор $B(N)$, элементы B_i которого равны единице, если элементы i -й строки образуют упорядоченную по убыванию или по возрастанию последовательность, и нулю во всех остальных случаях.

86. Определите, сколько строк заданной матрицы $A(N,M)$ содержат хотя бы один элемент из заданного числового диапазона.

87. Найдите номера строк заданной целочисленной матрицы, в которых:
а) на всех нечетных позициях стоят нули; б) на нечетных позициях встречаются нули.

88. Подсчитайте количество различных (не повторяющихся) чисел, встречающихся в заданном целочисленном массиве $A(N)$. Повторяющиеся числа учитывайте только один раз.

89. Подсчитайте количество различных (не повторяющихся) чисел, встречающихся в заданной целочисленной матрице $A(N,M)$.

90. Даны сведения о количестве голов, забитых каждым футболистом команды в каждом из матчей чемпионата. Проверьте, сколько в команде есть футболистов: а) забивших хотя бы два гола; б) забивавших голы в каждом матче; в) не забивших ни одного гола.

91. Используя сведения о ежемесячных выплатах зарплаты сотрудникам фирмы, выясните, не оказалась ли годовая зарплата кого-либо из сотрудников ниже годового минимума, оговоренного в его контракте.

92. Используя сведения о результатах сдачи n вступительных экзаменов m абитуриентами, определите, сколько абитуриентов сдали все экзамены на "отлично".

93. Найдите максимальное из чисел, встречающихся в заданной матрице более одного раза.

94. Подсчитайте количество запятых в заданном тексте.

95. Подсчитайте, сколько раз в заданном тексте встречается заданный символ.

96. Определите долю пробелов в заданной строке.

97. Проверьте, является ли заданное слово названием времени года на русском языке.

98. Замените в заданном тексте буквосочетание "min" на "max".

99. В заданном тексте подсчитайте общее количество букв "а" и "о".

100. Найдите самое длинное и самое короткое слово в заданном предложении.

Заключение

Современное общество живет в период, характеризующийся небывалым ростом объема информационных потоков. Это относится как к экономике, так и к социальной сфере. Наибольший объем информации наблюдается в промышленности, торговле, финансово-банковской деятельности. В промышленности рост объема информации обусловлен увеличением объема производства, усложнением выпускаемой продукции, используемых материалов, технологического оборудования, расширением в результате концентрации и специализации производства внешних и внутренних связей экономических объектов. В этой связи предъявляются повышенные требования к алгоритмам обработки информации.

В учебном пособии осуществляется пропедевтика понятий "алгоритм", "исполнитель" и др., раскрываются подходы к разработке алгоритмов, рассматривается большое количество примеров, позволяющее повысить качество изучения представленного материала. Показано, что процесс алгоритмизации любой задачи, достаточно сложный, многоэтапный процесс.

Представленный материал является основой для дисциплин, связанных с программированием, моделированием, передачей и обработкой информационных потоков данных.

Библиографический список

Основной

1. Шауцукова Л. З. Информатика: Учеб. пособие для 10 – 11 кл. общеобразоват. учреждений/ Л.З. Шауцукова. – М.: Просвещение, 2000.
2. ГОСТ 19.701–90 (ИСО 5807 – 85) "Единая система программной документации".
3. Островейковский В. А. Информатика: Учебник для вузов. – М.: Высш. шк., 2000.
4. Николаев В. И., Чалов Д. В., Сибирев В. Н. Информатика. Теоретические основы: Учеб. пособие. – СПб.: СЗТУ, 2002.
5. Николаев В. И., Иванова И. В. Теория алгоритмов: Текст лекций. – СПб.: СЗТУ, 1995.

Дополнительный

6. Кушниренко А. Г. и др. Информатика.– М.: Дрофа, 1998.
7. Кулаков А. Г., Ландо С. К., Семенов А. Л., Шень А. Х. Алгоритмика, V–VII классы. – М.: Дрофа, 1996.
8. Кузнецов А. А. и др. Основы информатики. – М.: Дрофа, 1998.
9. Гейн А. Г., Сенокосов А. И. Информатика. – М.: Дрофа, 1998.
10. Ландо С. К. Алгоритмика: Методическое пособие. – М.: Дрофа, 1997.
11. Зотов В. В. и др. Терминологический словарь по автоматике, информатике и вычислительной технике. – М.: Высшая школа, 1989.
12. Борцовский Г. А. Информатика в понятиях и терминах, – М.: Просвещение, 1991.
13. Вычислительная техника и программирование/ Под ред. А. В. Ретрова. – М.: Высшая школа, 1990.
14. Боглаев Ю. П. Вычислительная математика и программирование. – М.: Высшая школа, 1990.
15. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985.
16. Ван Тассел Д. Стил, разработка, эффективность, отладка и испытание программ.– М.: Мир, 1981.
17. Алексеев А. Ю., Ивановский С. А., Куликов Д. В. Динамические структуры данных. Практикум по программированию/ ГЭТУ. СПб., 2000. – 76 С.

Предметный указатель

А

Алгоритмизация 5

Алгоритм 5, 20

 разветвляющийся 31

 циклический 31

Алгоритмы

 механические 6

 гибкие 6

 вероятностный 6

 эвристический 6

Алфавит 19, 27

Алгоритмический язык 25

Арифметические выражения 29, 30

Б

Блок-схема 11

В

Величины 21

Вещественные числа 21

Выражения 22, 24, 29

Внутренний цикл 35

Внешний цикл 35

Вспомогательные алгоритмы 38

Г

Графы 17

 сигнальные 17

 автоматные 17

Д

Декомпозиция 7

Диаграммы 16

Данные 28

З

Зона 14

Записи 15

Заголовок 20

Знаки действий 22

И

Исполнители алгоритмов 25

Итерационный цикл 32

К

Координаты зоны 14

Ключевые слова 19

Комментарии 21

Л

Литерные значения 21

Логические значения 21

Линейный список 28

Логические выражения 30

М

Математическое обеспечение 25

Массивы 28

Метод решения 41, 42

Метод частных целей 44

Метод подъема 45

Н

Натуральные числа 21

О

Описание алгоритмов

словесный 10

графический 11

Описание типа данных 21

Операторы 22, 29

- простые 22
- составные 22
- присваивания 23

Описание массивов 23

Операции 28

- арифметические 28
- логические 28
- отношений 28

П

Псевдокод 18

Переменные 23, 28

Программный способ 25

Принципы написания программ 30

Постановка задачи 42

Правила составления алгоритма 43

Р

Результат 19, 22, 39

С

Свойства алгоритмов

- понятность 8
- дискретность 8
- определенность 8
- результативность 8
- массовость 9

Схема алгоритма 11

Символ 11, 14, 16

- "Процесс" 11

- "Решение" 12

- "Модификация" 13

- "Предопределенный процесс" 13

- "Документ" 14

- "Ввод - вывод" 14

- "Соединитель" 14

"Пуск - останов" 14

"Комментарий" 14

Схемы блокировки 16

Структурные схемы 17

Сети Петри 17

Схемы работы 18

Синтаксические конструкции 21

Символьные значения 21

Семантика 27

Синтаксис 27

Структуры алгоритмов 31

Стандартные функции 39

Сценарий 42

Т

Тело алгоритма 20

Тип значения 20

У

Утверждения 21

Указатель функции 39

Ф

Фактические параметры 39

Функция 39, 40

Х

Характеристики алгоритмов

временные 9

объемные 9

Характеристики 20

Э

Эвристика 6

Этапы решения 7, 41

Я

Языки 25, 26

высокого уровня 26

логические 26

машинные 26

машинно-ориентированные 26

объектно-ориентированные 26

процедурные 26

Оглавление

Введение	3
Глава 1. Алгоритмические основы программирования	5
1.1. Понятие алгоритма	5
1.2. Свойства алгоритмов	8
1.3. Основные характеристики алгоритмов	9
1.4. Способы описания алгоритмов	10
1.4.1. Словесный способ представления алгоритмов	10
1.4.2. Графический способ представления алгоритмов ...	11
1.4.2.1. Правила выполнения схем	14
1.4.2.2. Соотношение геометрических элементов симво- лов.....	16
1.4.2.3. Графические способы описания алгоритмов ра- боты информационных систем (промышленных систем).....	16
1.4.3. Псевдокоды	18
1.4.4. Программный способ представления алгоритмов ...	25
1.4.4.1. Достоинства и недостатки машинных языков	26
1.4.4.2. Компоненты алгоритмического языка	27
1.4.4.3. Понятия, используемые в алгоритмических языках .	28
1.5. Структуры алгоритмов	31
1.6. Вспомогательные алгоритмы	38
1.7. Технология решения задач с использованием компьютера	41
1.7.1. Основные этапы решения задач с помощью компьютера	41
1.7.2. Приемы алгоритмизации расчетных задач	44
1.7.2.1. Метод частных целей	44
1.7.2.2. Метод подъема	45
1.7.2.3. Программирование с отходом назад	45

1.7.2.4. Алгоритмы ветвей и границ	45
Вопросы для самопроверки	46
Глава 2. Практикум по алгоритмизации и программированию ...	47
2.1. Алгоритмы линейной структуры	47
2.2. Алгоритмы с разветвляющейся структурой	49
2.2.1. Инструкция если (<i>if</i>)	49
2.2.2. Инструкция выбор (<i>switch</i>)	53
2.3. Задачи для самостоятельного решения	58
2.4. Алгоритмы, реализуемые с помощью циклов	59
2.4.1. Инструкция для (<i>for</i>)	59
2.4.2. Инструкция выполнять пока (<i>do-while</i>)	62
2.4.3. Инструкция пока (<i>while</i>)	66
2.5. Задачи для самостоятельного решения	70
Заключение	78
Библиографический список	79
Предметный указатель	80

Михаил Петрович Белов

Основы алгоритмизации в информационных системах

Учебное пособие

Редактор

Сводный темплан 2003 г.

Лицензия ЛР № 020308 от 14.02.97

Подписано в печать	Формат 60 x 84 1/16
Б.кн.-журн. П.л. 5,3. Б.л.	РТП РИО СЗТУ.
5,3.	Заказ
Тираж 400.	

Северо-Западный государственный заочный технический
университет
РИО СЗТУ, член Издательско-полиграфической ассоциации
вузов Санкт-Петербурга
191186, Санкт-Петербург, ул. Миллионная, 5