1. АЛГОРИТМЫ

1.1. Понятие алгоритма

Представьте себе, что Вам необходимо объяснить ученику 6-го класса, как складывать дроби. Поскольку преподавать математическую теорию вопроса не хочется, Вы решаете представить метод решения задачи в виде последовательности действий над числами, которые он должен выполнять, не вникая в суть дела. Это и есть алгоритм решения задачи.

Термин «алгоритм» впервые был использован средневековыми учеными, которые переводили на латынь произведения узбекского ученого Аль Хорезми. Алгоритмами они называли правила арифметических действий над многоразрядными числами.

Такой метод изложения способа решения задач широко распространен. В виде алгоритмов выкладывают кулинарные рецепты, правила пользования бытовыми приборами, служебные инструкции и т.д. В информатике алгоритмы используются для описания способа решения задач по преобразованию информации.

Точное математическое определение алгоритма и изучение этого понятия - предмет спе-циальной математической дисциплины - теории алгоритмов, которая, в свою очередь, опирается на аппарат математической логики.

Здесь мы рассмотрим на содержательном (неформальном) уров-не лишь характерные основные черты этого понятия.

Алгоритм - это некоторое правило преобразования информации, применение которо-

го к заданной (исходной) информации приводит к резуль-тату - новой информации.

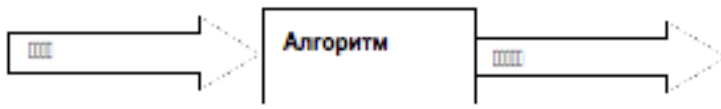


Рис. 1.1 Схема алгоритма

Так, например, применение правила сложения дробей к 1/2 и 2/3 приводит к результату 5/6.

Основное внимание в теории алгоритмов уделяется методам задания (описания, конструи-рования) алгоритмов. Один из основных подходов к описанию алгоритмов - так называемый императивный подход, который заключается в уточнении способа описания алгоритма как последовательности шагов, на каждом из которых выполняется одна из ко-

манд (инструкций, операторов).

Алгоритм - это конечный набор инструкций по преобразованию информации (команд), выполнение которых приводит к результату. Каждая инструкция алгоритма содержит точное описание некоторого эле-ментарного действия по преобразованию информации, а также (в явном или неявном виде) указание на инструкцию, которую необходимо выполнить следующей.

На рис. 1.2 изображена последовательность команд C1, C2, ..., Ck по преобразованию информации, с которых составлен алгоритм.

Вспомним древнюю задачу-головоломку про волка, козу и капусту, которые во главе с человеком вышли к реке и должны переправиться на другой берег. В распоряжении этой

веселой компании есть лодка, содержащая вместе с человеком еще только одного ее члена - или волка, или козу, или капусту. Если волк окажется наедине с козой, он ее съест. Если коза окажется наедине с капустой, она ее (капусту) съест. Требуется составить такой план перевозок, при котором наша компания переберется на другой берег в целости и сохранности. Этот план и есть алгоритм. Опишем его:

Алгоритм Перевозки:

1. Перевести козу.

2. Вернутся.

3. Перевести волка.

4. Перевести козу.

5. Перевести капусту.

6. Вернутся.

7. Перевести козу.

Обратите внимание на следующие обстоятельства. Для того, чтобы найти этот план-алгоритм, нужно немного подумать (на то она и головоломка). Нужно также обосновать правильность этого плана, иначе кого-нибудь съедят. Наконец, нужно так оформить решение, чтобы его смог понять и выполнить перевозчик – то есть исполнитель алгоритма.

Это и есть те основные проблемы, с которыми постоянно имеет дело программист, разрабатывая программы для компьютеров.

Алгоритм сложения дробей можно задать следующей последовательностью команд:

Пример 1. Алгоритм сложения дробей.

Вход: A/B, C/D;

1. Вычислить Y = B*D; {Перейти к следующей команде}

2. Вычислить X_1 = A*D; {Перейти к следующей команде}

3. Вычислить X_2 = B*C; {Перейти к следующей команде}

4. Вычислить X = X_1+X_2; {Перейти к следующей команде}

5. Вычислить Z = НОД(X,Y); {Перей-

ти к следующей команде}

6. Вычислить E = X div Z; {Перейти к следующей команде}

7. Вычислить F = Y div Z; {Закончить работу}.

Выход: E/F

Исходная информация этого алгоритма представлена четырьмя целыми числами A, B, C, D. Это - числитель и знаменатель слагаемых. Результат работы алгоритма - числа E и F - числитель и знаменатель суммы.

Информацию, исходную для алгоритма, принято называть его входом, а результат выполнения - выходом.

Знать, какую задачу решает алгоритм, необходимо только тому, кто хочет использовать этот алгоритм для своих целей. Таким образом, первые две строки примера 1 содержат информацию, необходимую пользователю алгоритма. Исполнитель алгоритма ничего не знает и не должен знать о том, какую задачу он решает. В этом и заключается свойство формальности выполнения алгоритма.

Наш алгоритм состоит из 7-ми инструкций, каждая из которых содержит описание одного из арифметических действий над целыми числами: сложение, умножение, вычисление НОД и целочисленное деление div. Кроме того, каждая инструкция (в неявном виде) содержит указание на следующую выполняемую инструкцию.

Таким образом, алгоритм описывает детализированный по шагам процесс преобразования информации. Исполнитель алгоритма

не только выполняет действия, но и запоминает их результаты. Для отражения этого факта в записи алгоритма мы используем буквенные обозначения данных. Эти обозначения называют именами, а сами данные - величинами.

Для обозначения величин в алгоритмах используются имена.

Последовательный порядок выполнения команд необходим только для того, чтобы результаты выполнения предыдущих команд можно было бы использовать как исходные данные в выполняемой команде.

В нашем примере команды 1, 2, 3 вычисления величин X1, X2, Y можно выполнять или параллельно (в три руки), или изменять порядок их выполнения. От этого результат алгоритма не изменится. Это же можно сказать о командах 6 и 7. Ответ на вопрос о том,

представляют ли эти варианты записи алгоритма тот же алгоритм, или нет, зависит от точного определения понятия алгоритма. Существенным является то обстоятельство, что две формы записи могут представлять алгоритмы, которые дают тот же результат.

Отметим еще одно важное свойство алгоритма - он рассчитан на исполнителя, который понимает и умеет выполнять команды алгоритма.

Ученик 6-го класса, скорее всего, умеет составлять, умножать и делить уголком целые числа. Однако не все дети умеют находить НОД - наибольший общий делитель двух целых чисел. Это означает, что они не смогут выполнить Ваш алгоритм.

Уровень детализации описания опре-де-ляется набором допустимых команд. Этот набор называют набором команд Ис-

полнителя или Интерпретатора.

1.2. Исполнитель алгоритмов и его система команд

При этом подразумевается, что алгоритмы выполняет Исполнитель (Интерпретатор) - некоторое устройство, однозначно распозна-ющее и точно выполняющее (интерпретирующее) каждую команду алгоритма.

Для выполнения нашего алгоритма Исполнитель должен, очевидно, уметь оперировать с целыми числами, выделять числители и знаменатели дробей, а также составлять из пары целых чисел дробь. Кроме того, Исполнитель должен уметь запоминать результаты выполнения опера-ций и переходить к выполнению следующей команды.

Представим себе, что в нашем распоряжении находится Исполнитель, интерпретиру-

ющий команды - операции целочисленной арифметики - сложение, вычитание, умножение, вычисление неполного частного (div) и остатка (mod), вычисление НОД и НОК с запоминанием результатов и умением переходить к следующей команде. Тогда для этого Исполнителя можно составлять самые разнообразные алгоритмы арифметических вычислений - т.е. вычислений, заданных формулами типа

X = НСД((A + B) div 100, (A*B - 7) mod 10),

используя команды, аналогичные командам алгоритма из примера 1.1. Для этого необходимо, учитывая приоритеты арифметических операций, правильно определить последовательность выполнения арифметических действий и записать ее в виде последовательности команд.

Пример 1.2 Алгоритм деления отрезка пополам с помощью циркуля и линейки.

Вход: Отрезок AB;

Построить окружность O_1 с центром A и радиусом AB;

Построить окружность O_2 с центром B и радиусом AB;

Найти точки C и D пересечения окружностей O_1 и O_2;

Построить отрезок CD;

Найти точку E пересечения AB и CD.

Выход: Точка E - середина отрезка AB.

В примере 1.2 Исполнитель обладает набором команд, с помощью которых можно решать геометрические задачи на построения с помощью циркуля и линейки. Назовем этого Исполнителя Геометром. Исполнение алгоритма заключается в последовательном выполнении каждого построения и переходе к исполнению следующей команды.

Отметим. что нумеровать команды алгоритма не нужно. Перечислим команды Геометра:

Построить отрезок s с концами в точках A, B:

Вход: точки A, B.

Выход: отрезок s с концами в точках A, B.

Построить прямую l через точки A, B:

Вход: точки A, B.

Выход: прямая l, которая проходит через точки A, B.

Построить круг O с центром A и радиусом BC:

Вход: точки A, B, C.

Выход: круг O с центром A и радиусом BC.

Найти точку A пересечения прямых l_1 и l_2 :

Вход: прямые l_1, l_2.

Выход: точка A пересечения l_1 и l_2.

Найти точки A и B пересечения круга O и прямой l;

Вход: коло O, прямая l.

Выход: точки A и B пересечения круга O и прямой l.

Найти точки A и B пересечения кругов O_1 и O_2;

Вход: круги O_1, O_2.

Выход: точки A и B пересечения кругов O_1 и O_2.

В отличие от примера 1.1, в примере 1.2 величинами являются простейшие геометрические фигуры - точки, прямые, окружности. Они служат и выходными данными, и результатами команд. Каждая из команд исполнителя Геометр выполняет одно из действий, которые можно выполнить с помощью циркуля или линейки.

Таким образом, набор команд исполнителя Геометр ориентирован на решение точно

определенного класса задач - геометрических задач на построения с помощью циркуля и линейки.

Описание каждой команды включает в себя:

· точный «внешний вид»,

· входные величины,

· выходные величины,

· связь между входными и выходными данными.

Совокупность величин, рассмотренная вместе с набором допустимых преобразований, образует предметную область.

Исполнитель алгоритмов умеет выполнять команды, каждая из которых опреде-

ляет одно из допустимых преобразований. Алгоритм является последовательностью этих команд. Поэтому исполнитель предназначен для выполнения любого алгоритма над данной предметной областью.

Приведем еще несколько примеров предметных областей.

1. Шахматы. Эта предметная область представляет собой шахматную доску, на которой расположены белые и черные шахматные фигуры. Каждая из фигур описывается совокупностью своих допустимых ходов и правилами взаимодействия с другими фигурами. Поэтому в шахматах возможны постановки алгоритмических задач.

Например: Построить алгоритм, которые переводит фигуру «конь» с поля a1 на поле h8. Других фигур на шахматной доске нет.

2. Дороги города. Эта предметная область представлена картой города, на которой обозначены улицы и перекрестки, а также списком видов транспортных средств, которые ездят по городу. Определены правила движения вдоль каждой улицы, разрешающие или запрещающие движение вдоль улицы в данном направлении тех или иных видов транспорта, а также аналогичные правила проезда через перекресток. В этой ситуации возможны постановки алгоритмических задач.

Например Проехать на легковом автомобиле от одного перекрестка к другому через минимально возможное число промежуточных перекрестков.

3. Черепашка. Алгоритмический язык Лого, предназначена для обучения основам алгоритмизации младших школьников, исполнителем алгоритмов выступает Черепаш-

ка. Черепашка умеет двигаться в разных направлениях, задаваемых в командах, рисовать хвостом линии вдоль направления своего движения. Программист имеет возможность описывать алгоритмы рисования простых картинок.

Например: Нарисовать домик с крышей, одной дверью и двумя окнами.

1.3. Основные свойства алгоритмов

Понятию алгоритма присущи следующие свойства:

1. Элементарность. *Каждая команда из набора команд Исполнителя содержит указание выполнить некоторое элементарное (не детализируемое более подробно) действие, однозначно понимаемое и точно выполняемое Исполнителем.*

Понятие элементарности поэтому и относительно. Так, в алгоритме примера 1 содержится команда вычисления НОД двух чисел. Это означает, что Исполнитель умеет находить НОД, причем алгоритм вычисления (алгоритм Евклида или какой- нибудь другой) скрыт от человека, составляющего алгоритмы для этого Исполнителя. Если набор команд Исполнителя не содержит команды вычисления НОД, вычисление НОД должно быть определено в виде алгоритма.

Пример 1.3 Алгоритм Евклида вычисления наибольшего общего делителя целых положительных чисел A и B: НОД(A, B).

Вход A, B;

Вычислить U = A;

Вычислить V = B;

Пока U <> V выполнять

Если U < V

то Вычислить V = V - U

иначе Вычислить U = U - V;

Вычислить D = U.

Выход: D - наибольший общий делитель A и B.

В этом примере использована команда повторения.Она имеет вид

Пока <Условие> **выполнять** <Команда>

Выполняя эту команду, Исполнитель проверяет истинность Условия. Если Усло-вие истин-но, Исполнитель выполняет Команду, указанную после слова выпол-нять и повторяет проверку Условия. Выполнение Команды и проверка Условия повторяются до тех пор, пока Условие истин--но. Если Условие ложно, Исполнитель переходит к выполнению команды, следующей за командой пов-то-рения. В этом же примере использована еще одна разновидность команды ветвления - команда вида

Если <Условие> **то** <Команда 1> **иначе** < Команда 2>

Выполняя эту команду, Исполнитель проверяет Условие. Если Условие выполнено, выпол-няется Команда 1, в противном случае выполняется команда 2. Далее Исполнитель

переходит к сле-дующей команде. Заметим, что команда повторения, как и команды ветвления, содержат в себе другие команды.

2.Определенность. *Исполнение алгоритма строго определено. Это означает, что на каждом шаге Исполнитель не только точно выполняет команду, но и однозначно определяет следующую исполняемую команду. Поэтому повторное выполнение алгоритма для одних и тех же исходных данных в точности повторяет первое его выполнение.*

Так, в исполнении алгоритма в примере 1.3 возможны ветвления, которые, однако, одно-значно определены условиями D < 0, D = 0.

3.Массовость. *Алгоритмы, как правило, описывают ход решения не одной-единственной задачи, а целого класса однотип-*

ных задач.

Так, в примере 1.1 описан алгоритм сложения любых двух дробей. Одна-единственная задача, решаемая Исполнителем в данный момент, определена значениями исходных данных A, B, C, D. Изменение исходных данных означает решение другой задачи из этого же класса задач.

Аналогично, алгоритм примера 1.2 строит середину любого отрезка, заданного его концами, а в примере 3 с помощью алгоритма решается любое приведенное квадратное уравнение.

4. Результативность. *Исполнение любого алгоритма должно быть закончено через конеч-ное число шагов (т.е. выполнение конечного числа команд) с некоторым результатом.*

Так, в примере 1.2 результат - точка E - середина отрезка AB. Алгоритм примера 1.5 выдает результат "Решений нет" даже в том случае, когда уравнение не имеет действительных корней, поскольку его дискриминант меньше нуля.

Отметим, однако, что количество шагов алгоритма, который решает некоторую задачу, заранее неизвестна и может быть очень большой, поэтому свойство результативности конкретного алгоритма часто приходится специально доказывать. Программист должен быть уверенным в том, что составленный им алгоритм, во-первых, всегда завершает работу, и во-вторых - работает правильно, т.е. выдает правильный ответ.

Для обоснования правильности алгоритма Евклида нужно доказать, что команда повторения всегда завершается и d = НСД(a, b).

1.4. Величины

Величиной называют такую характеристику предмета или явления, значение которой можно измерить или вычислить.

В курсе физики Вы изучали закон Ома для участка электрической сети, связывающей три физических величины I, U, R равенством

$$U = I*R.$$

Здесь I - сила тока, протекающего через резистор, U - падение напряжения на этом резисторе, R - сопротивление резистора.

Для физика закон Ома - формула, которую можно проверить экспериментально, а также использовать для вычисления одной из величин по двум другим.

Для программиста, который решает задачу построения алгоритма вычисления U при заданных значениях I и R, дело обстоит иначе. Во-первых, он должен определить такую форму представления величин U, I, R, которая пригодна для Исполнителя, во-вторых - представить операцию умножения как алгоритм в системе команд Исполнителя. Физический смысл величин алгоритма для программиста никакой роли не играет.

Алгоритм определяется не только задачей, но и системой команд Исполнителя.

Представим себе, что в вашем распоряжении есть устройство, выполняющее арифметические операции с целыми числами, представленными в десятичной системе счисления. Заказчик алгоритма (физик) определил форму представления входных данных и результата в привычном для себя виде: каждая из величин I, R представлена в виде десятич-

ной дроби с порядком, т.е. в форме

<целая часть>.<дробная часть>*10$^{<порядок>}$

где целая часть, дробная часть и порядок записаны в десятичной системе счисления.

Пример входных данных: I = 34.23*10^{-2}, R = 581.6*10^{3}.

Такая форма представления действительного числа называется формой с плавающей точкой.

Результат U нужно представить в таком виде. Определение единиц измерения (вольты, амперы, омы) физик берет на себя.

Ваш алгоритм должен:

· вычислить целую часть произведения;

· вычислить дробную часть произведения;

· вычислить порядок произведения.

1.5. Типы величин

Представление о величинах прошло долгий исторический путь развития. Безусловно, на практике наиболее распространены числовые величины, значениями которых являются числа.

В глубокой древности люди стали пользоваться натуральными числами. Понятие натурального числа сформировалось как результат абстрагирования в задачах определения количества предметов или местоположения данного предмета в ряду других.

Рациональные числа появились в задачах распределения предмета на равное число частей. Положительными рациональными чис-

лами пользовались еще Евклид и Пифагор.

Десятичная система счисления пришла в средневековую Европу вместе с арабскими цифрами и вскоре стала общепринятой.

Значительно позже были введены в употребление число 0 и отрицательные числа. В результате оформилось понятие целого числа.

Действительные числа стали использоваться для приближенных измерений и вычислений.

Бурное развитие математики как науки о числах привело к открытию алгебраических и трансцендентных чисел, комплексных чисел и многих других числовых систем.

В результате развития геометрии, механики, других наук сформировалось представле-

ние о векторных величинах. В отличие от скалярной величины, значение которой определяется одним числом, векторная величина определяется несколькими числовыми значениями.

Так, положение точки на плоскости определяется парой ее координат, а скорость движения, кроме величины, определяется еще и направлением. При движении точки в пространстве ее положение определяется тремя пространственными координатами и временной координатой - временем.

Представление о строчных величинах сформировалось в процессе становления информатики как науки. Значением строчной величины есть слово, есть цепочка букв из некоторого алфавита.

Примеры буквенных данных: 'Информатика', 'Algorithm', '5 мая 2004 ',' триста два-

дцать девять ". Необходимость рассматривать слова как данные возникает в алгоритмах обработки текстовой информации.

Еще один пример величин, не являющихся числовыми - так называемые логические величины. Логическая величина может принимать два логических значения - Истина и Ложь. Подробнее этот тип величин мы рассмотрим позже.

Наконец, значениями величин из примера 1.2 является простейшие геометрические фигуры. Это пример величин, которые можно называть геометрическими.

Величины, используемые в алгоритме, характеризуется именем, типом и значением.

Имя величины идентифицирует эту величину. Программист использует имена для обозначения величин. Исполнитель алгорит-

ма получает доступ к данной величины по ее имени.

Тип величины определяет набор допустимых операций над данной величиной, область ее определения и форму записи ее значений.

Каждая величина, используемая в алгоритме, в каждый момент времени имеет некоторое значение. Значением величины есть данное соответствующего типа.

Во многих современных алгоритмических языках принято описывать все величины, используемые в алгоритме. Описание величины включает ее имя и тип. Поскольку в алгоритмах, кроме входных и выходных величин, используются еще и вспомогательные величины, их тоже нужно описывать.

Вернемся к алгоритму примера 1.1. Запи-

шем его по всем правилам, принятым ранее.

Приклад 1.1 Продолжение

Алгоритм Сложение дробей;

Вход

A, C: Целые числа;

B, D: Натуральные числа;

Выход

E: Целое число;

F: Натуральное число;

Дополнительные величины

Y, Z: Натуральные числа;

X, X_1, X_2, E: Целые числа;

Начало

Вычислить Y = B*D;

Вычислить X_1 = A*D;

Вычислить X_2 = B*C;

Вычислить X = X_1+X_2;

Вычислить Z = НСД(X,Y);

Вычислить E = X div Z;

Вычислить F = Y div Z

Конец.

В математике принято считать, что знаменатель рационального числа, записанного в виде дроби, всегда положительный. Поэтому, определяя типы величин, мы классифицируем величины B и D, а также вспомогательные величины Y, Z как натуральные числа.

1.6. Целые числа

Величины, представлены целыми числами, используются практически в каждом алгоритме. Поэтому мы рассмотрим сейчас те операции с целыми числами, которые принято считать элементарными.

Арифметические операции:

a + b – операция сложения

a - b – операция вычитания,

- b – операция «минус»;

a*b – операция умножения,

a div b – операция вычисления неполного частного,

a mod b – операция вычисления остатка.

Логические операции:

a > b - операция «больше»

a < b - операциія «меньше»

a <= b - операция «меньше либо равно»

a >= b - операция «больше либо равно»

a = b - операция «равно»

a <> b - операция «не равно»

Арифметические операции сложения, вычитания и «минус» называют аддитивными, а операции умножения, вычисления неполного частного и остатка - мультипликативными.

Мы будем считать, что операции div и mod правильно интерпретируются только тогда, когда оба операнда - натуральные числа. Для этих операций справедливо соотношение

A = B * (A div B) + A mod B.

причем

A div B >= 0, 0 <= A mod B < B.

Это означает, что любое натуральное чис-

ло A можно разделить на другое натуральное число B с остатком, причем неполное частное неотъемлемо, а остаток может принимать значения 0, 1, ..., B-1.

Например, если A = 56, B = 9, то A div B = 6, A mod B = 2, и 56 = 9 * 6 + 2.

В следующей задачи мы используем операции div и mod для того, чтобы выделить цифры числа.

Задача 1.1 Составить алгоритм, который переставляет местами старшую и младшую цифры данного трехзначного десятичного числа ABC, т.е. строит число CBA.

Решение.

Обозначим через A, B и C - соответственно старшую, среднюю и младшую цифры входного числа ABC, а через AB - двузначное чис-

ло, полученное из ABC отбрасыванием последней цифры. Тогда

C = ABC mod 10, AB = ABC div 10.

Мы «расщепили» число на младшую цифру и число, составленное из старших цифр. С числом AB делаем то же самое:

B = AB mod 10, A = AB div 10.

Осталось "собрать" число CBA с его цифр A, B, C:

CBA = 100*C + 10*B + A.

Алгоритм Переверни число;

Вход

ABC : Натуральное число;

Выход

BCA : Целое число;

Дополнительные величины

A, B, C, AB: Целые числа;

Начало

Вычислить C = ABC mod 10;

Вычислить AB = ABC div 10;

Вычислить B = AB mod 10;

Вычислить A = AB div 10;

Вычислить CBA = 100*C + 10*B + A

Конец.

Задача 1.2 Банкомат заряженный купюрами номиналом 100, 50, 20, 10, 5, 2, 1. Составить алгоритм, определяющий, сколько купюр каждого номинала нужно, чтобы выдать сумму в N денежных единиц. Ваш алгоритм должен тратить при этом наименьшее возможное количество купюр.

Решение.

Дано: натуральное число N. Обозначим через K100 количество сотенных купюр, K50 - количество полтиноков, и т.д. Наш алгоритм будет подобным жадному человеку: на каждом шагу он будет определять и вычитать из суммы максимально возможное количество купюр данного номинала, начиная с самых крупных.

В задачах программирования очень часто требуется найти оптимальное решение. «Жадные» алгоритмы стремятся на каждом шагу найти «максимальную часть» такого решения. Эта стратегия часто, хотя далеко не всегда, приводит к оптимальным решениям.

Алгоритм Банкомат;

Вход

N: Натуральное число;

Выход

K100, K50, K20, K10, K5, K2, K1: Целые числа;

Начало

Вычислить K100 = N div 100; {Количество сотен**}**

Вычислить N = N mod 100; {Остаток N < 100**}**

Вычислить K50 = N div 50; {Количество полтинников**}**

Вычислить N = N mod 50; {Остаток N < 50**}**

Вычислить K20 = N div 20; {Количество двадцаток**}**

Вычислить N = N mod 20; {Остаток N < 20**}**

Вычислить K10 = N div 10; {Количество десяток**}**

Вычислить N = N mod 10; {Остаток N < 10**}**

Вычислить K5 = N div 5; {Количество пятерок**}**

Вычислить N = N mod 5; {Остаток N < 5**}**

Вычислить K2 = N div 2; {Количество двоек**}**

Вычислить K1 = N mod 2 {Осталось единиц < 2**}**

Конец.

Задача 1.3 Составить алгоритм, который вычисляет целые коэффициенты A, B, C квадратного уравнения ax2 + Bx + C = 0 по

его рациональными корнями x1 = n1 / m1, x2 = n2 / m2

Решение.

Дано: целые числа n1, m1, n2, m2. Нужно найти целые числа A, B, C. Связь между корнями и коэффициентами квадратного уравнения устанавливает теорема Виета: если x1 и x2 - корни приведенного квадратного уравнения x2 + px + q = 0, то x1 + x2 =-p, x1 * x2 = q. Поэтому

$n_1/m_1 + n_2/m_2 = - B/A$, $n_1/m_1 * n_2/m_2 = C/A$.

Отсюда $(n_1*m_2+n_2*m_1)/(m_1*m_2) = - B/A$ $(n_1*n_2)/(m_1*m_2) = C/A$.

Приравняем числитель и знаменатель дробей в этих равенствах:

$A = m_1*m_2$, $B = - (n_1*m_2+n_2*m_1)$, $C = n_1*n_2$.

Алгоритм Теорема Виета;

Вход

m1, m2, n1, n2: целые числа;

Выход

A, B, C: целые числа;

Начало

Вычислить A = m1*m2;

Вычислить B = -(n1*m2 + n2*m1);

Вычислить C = n1*n2

Конец.

Программист должен знать ту предметную область, задачу которой он решает. Метод решения нужно искать именно там.

1.7. Действительные числа

Действительные числа представляют значения различных величин при вычислениях. Этим и определяется их применения в алгоритмах решения разного рода вычислительных задач. К элементарным операциям исполнителя обычно относят арифметические и логические операции, а также операции вычисления некоторых математических функций.

Принципиально важной особенностью действительных чисел является то, что они представляют приближенные значения

величин и все операции над ними следует считать приближенными.

В записи алгоритмов принято использовать действительные числа в десятичной системе счисления. Допускается запись чисел как в виде десятичных дробей (форма представления с фиксированной точкой), так и в форме с плавающей точкой.

Арифметические операции:

a + b – операция сложения

a - b – операция вычитания,

- b – операция «минус»;

a*b – операция умножения,

a / b – операция деление.

Логические операции: (см. Целые числа).

Математические функции

Вообще говоря, задача приближенного вычисления значения некоторой математической функции (например, синуса) не является элементарной. Ее, как правило, нельзя решить с помощью алгоритма, подобного алгоритму добавления дробей, т.е. выполняя фиксированное число арифметических действий над целыми числами. Поэтому специализированный исполнитель, ориентированный на математические расчеты, "знает" алгоритмы приближенного вычисления некоторого набора математических функций. К их числу, как правило, входят такие элементар-

ные функции, как sin (x), cos (x), ln (x), arctg (x), exp (x), sqrt (x), функции округления, выделения целой и дробной части.

Рассмотрим несколько вычислительных алгоритмов.

Задача 1.4. Построить математическую модель и составить программу вычислений координат материальной точки, брошенной с начальной скоростью Vo в направлении вектора a = (Xo, Yo) в момент времени t.

Решение.

Как известно из физики, материальная точка M, брошенная в некотором направлении, двигалась бы равномерно и прямолинейно, если бы не сила тяжести, которая действует на точку и направлена вертикально

вниз. Поэтому

$$x = t*V_0*\mathrm{Cos}(Alfa)$$

$$y = t*V_0*\mathrm{Sin}(Alfa) - g*t^2/2$$

Проекции единичного вектора направления на координатные оси вычисляются по формулам:

$$\mathrm{Cos}(Alfa) = x_0/\mathrm{sqrt}(x_0^2 + y_0^2)$$

$$\mathrm{Sin}(Alfa) = y_0/\mathrm{sqrt}(x_0^2 + y_0^2)$$

Поэтому

$$x = V_0*t*x_0/\mathrm{sqrt}(x_0^2 + y_0^2)$$

$$y = V_0*t*y_0/\mathrm{sqrt}(x_0^2 + y_0^2) - g*t^2/2$$

Эти формулы решают математически поставленную задачу, т.е. представляют ее математическую модель. Программист, реализуя соответствующий алгоритм, должен позаботиться об экономии ресурсов исполнителя. Проще говоря, он так расписал вычисления, чтобы исполнитель не делал ту же работу дважды. В нашем случае это достигается предварительным вычислением значения выражения **Vo/sqrt (xo2 + yo2)**, входящий в каждую из формул.

$$A = V_o/\mathrm{sqrt}(x_o*x_o + y_o*y_o)$$

$$x = A*t*x_o$$

$$y = A*t*y_o - g*t^2/2$$

Следующее упрощение менее очевидно. Если в выражении для y вынести за скобки t, одной операцией умножения станет меньше.

$$A*t*y_0 - g*t^2/2 = (A*y_0 - g*t/2)*t$$

Забота об экономии ресурсов исполнителя приводит к следующей последовательности вычислений:

$$A = V_0/\mathrm{sqrt}(x_0*x_0 + y_0*y_0)$$

$$x = A*t*x_0$$

$$y = (A*y_0 - g*t/2)*t$$

Алгоритм Движение точки;

Вход

V_0, x_0, y_0, t: Действительные числа;

Выход

x, y: Действительные числа;

Константа

G = 9.8;

Дополнительная величина

A: Действительное число;

Начало

Вычислить A = V_0/sqrt(x_0*x_0 + y_0*y_0);

Вычислить x = A*t*x_0;

Вычислить y = (A*y_0 – g*t /2)*t

Конец.

В текст алгоритма включен раздел, в котором приведено имя и числительное значение константы всемирного тяготения. Константы являются вспомогательными величинами особого рода. Константы не изменяют своего значения в процессе выполнения алгоритма. Их имена и значения определяются в самом алгоритме. Команды алгоритма вместо значений констант используют их имена.

Задача 1.5. Составить алгоритм, который вычисляет первую цифру числа $\mathbf{a}^n$ (a - действительное число, n - натуральное число.)

Решение.

Дано: положительное действительное число a и натуральное число n. Найти: натуральное число z.

Идея решения заключается в следующем: для того, чтобы вычислить a^n, найдем десятичный логарифм числа a, умножим его на n, а затем пропотенцировать результат по основе 10.

$x = \lg(a)$, $y = n*x$, $z = 10^y$

Вспомним теперь, что целая часть числа y - это порядок числа z, а дробная часть, называемая также мантиссой, определяет значение z, т.е. его представление в виде последовательности цифр. Поэтому, если выделить дробную часть y и пропотенцировать результат, мы получим число z1, расположенное между 1 и 10 ($1 <= z1 <10$), первая цифра которого равна искомой.

x = lg(a), y = n*x, y = Frac(y), $z_1 = 10^y$, z = Int(z_1)

Алгоритм Первая цифра степени;

Вход

A: Действительное число;

N: Натуральное число;

Выход

Z: Натуральное число;

Дополнительная величина

y: Действительное число;

Начало

Вычислить y = lg(a);

Вычислить y = n*y;

Вычислить y = Frac(y);

Вычислить y = Pow(y, 10);

Вычислить z = Int(y)

Конец.

Составляя алгоритм, мы снова заботимся об экономии ресурсов исполнителя. На этот раз мы экономили память: величина y использовалась как место сохранения результатов промежуточных вычислений.

В алгоритме были использованы функции:

Lg(x) – десятичный логарифм x;

Frac(x) – дробная часть

Int(x) – целая часть

Pow(x, n) – x в степени n.

Система команд исполнителя должна быть снабжена алгоритмами вычисления этих функций.

Задача 1.6. Многочлены F (x) = a * x + b и G (x) = c * x + d заданы своими коэффициентами. Составить алгоритм вычисления коэффициентов многочлена H (x) = F (x) * G (x).

Решение.

Коэффициенты многочлена F (x) легко вычислить: перемножим двучлен F (x) и G (x) и сгруппируем коэффициенты при x2, x и свободный член.

(a*x + b)*(c*x + d) = (a*c)*x² +(b*c + a*d)*x +(b*d)

Отсюда **u = a*c, v = b*c + a*d, w = b*d.** Команды алгоритма:

Начало

Вычислить u = a*c,

Вычислить v = b*c + a*d,

Вычислить w = b*d.

Конец.

Исполнитель выполнит 4 умножения и 1 сложение. Заметим, что умножение - это более сложная операция, чем сложение. Для ее выполнения требуется больше времени. Поэтому для ускорения работы алгоритмов количество умножений в вычислениях стремят-

ся сделать минимальным.

Задача, которую мы решаем, является классическим примером неочевидной возможности такого ускорения. Оказывается, вычислить u, v, w можно, применив только 3 умножения.

Начало

Вычислить u = a*c,

Вычислить w = b*d,

Вычислить p = (a + b)*(c + d),

Вычислить v = p – u - w

Конец.

Одним умножением стало меньше за счет появления дополнительно 3-х аддитивных

операций.

1.8. Строковые величины

Строчные величины используются в алгоритмах обработки текстов. Типичный пример такого алгоритма - алгоритм поиска всех вхождений данного слова в текст и замены этого слова его синонимом.

Как уже упоминалось, значениями строчных величин есть слова, то есть цепочки символов (букв). Для того, чтобы уточнить систему команд исполнителя алгоритмов обработки текстов, нужно начать с алфавита, в символах которого записываются эти тексты.

За многовековую историю своего развития человечество изобрело много способов записи информации на материальных носителях (на камнях, глиняных табличках, бумаге, магнитном диске, в чипе памяти компью-

тера). В учебных пособиях по информатике упоминаются: текстовая информация, числовая информация, графическая информация, звуковая (аудио) информация, видеоинформация. Однако, независимо от вида информации, элементарной единицей ее представления является символ. Действительно, тексты состоят из букв и знаков. Числа мы записываем с помощью цифр и знаков-разделителей. В информатике графические образы (картинки) формируются из цветных точек, расположенных в прямоугольной таблице. Поэтому каждую точку представляют три числа: точка = (номер строки, столбца, номер цвета). Такое представление (кодирование) графической информации называют цифрованием. Цифрование информации - универсальный способ ее представления в компьютере.

Исполнитель текстовых алгоритмов располагает алфавит символов, из которых

должны состоять обрабатываемые им тексты.

Алфавит - это конечный упорядоченный набор символов. Символы расположены в алфавите в строго определенном порядке - один за другим.

Таким образом, в алфавите существует первый (начальный) символ и заключительный (конечный) символ. Для каждого символа алфавита, кроме заключительного, можно указать следующий за ним символ. Так же, для каждого символа алфавита, кроме начального, можно указать предшествующий ему символ.

Итак, символы алфавита можно сравнивать с помощью операций сравнения:

$s_1 < s_2$, $s_1 > s_2$, $s_1 <= s_2$, $s_1 >= s_2$, $s_1 = s_2$, $s_1 <> s_2$.

Например, для алгоритмов обработки

слов, представляющих десятичные дроби, алфавит состоит из следующих 13-ти символов:

{ 0 1 2 3 4 5 6 7 8 9 . - + }

Для того, чтобы отличать символы от других обозначений, в которых используются буквы, символы берут в кавычки.

В алфавите начальный символ - '0 ', а заключительный - знак "+". Символы нашего алфавита удовлетворяют цепочке неравенств:

'0' < '1' < '2' < '3' < '4' < '5' < '6' < '7' < '8' < '9' < '.' < '-' < '+'

Алгоритмы обработки текстов на украинском языке требуют гораздо больше символов. Это - прописные и строчные буквы алфавита, знаки препинания, скобки, другие символы, встречающиеся в текстах.

Для алфавита символов, используемых в информатике, существуют международные стандарты.

Словом называется конечная последовательность (цепочка) символов из алфавита исполнителя. Слова являются значениями строчных величин.

Наряду с термином "слово" в программировании используется также термины "строка", "литерал". (Термин "текст", которым мы пользовались, в программировании имеет несколько иное значение.)

Операции сравнения

Порядок, определенный на символах алфавита, используют для определения поряд-

ка на множестве строк. Этот способ сравнения и упорядочения строк (слов) называют алфавитным или лексикографическим порядком. Алфавитный порядок применяют при составлении словарей, справочников и т.д.

Для того, чтобы определить, какое из слов меньше, сравнивают сначала первые буквы этих слов, затем вторые, и так далее. Предположим, что сравниваются слова P = ‘$a_1a_2...a_k$‘ i Q = ‘$b_1b_2...b_m$’...

Если ‘a_1‘< ‘b_1‘ то P < Q;

Если ‘a_1‘ > ‘b_1‘ то P > Q;

Если ‘a_1‘= ‘b_1‘ то сравниваем символы ‘a_2 ‘ i ‘b_2’ , и так дальше.

На каждом шаге сравнения возможны следующие ситуации:

Одно из слов уже прочитано целиком, а второе - еще нет. В этом случае меньше то слово, символы которого уже прочитаны.

Оба слова прочитаны полностью. В этом случае слова равны.

Символ одного из слов меньше, чем символ другого. В этом случае меньше то слово, символ которого меньше.

Символы обоих слов равны. В этом случае переходим к сравнению следующих символов.

Например:

'мама' < 'папа', поскольку 'м' < 'п'.

'алгебра' < 'алгоритм', поскольку первые три буквы равны, а 'е' < 'о'.

‘program’ < ‘programmer’, поскольку при буквенном сравнении в первом слове все буквы будут исчерпаны, а в другом еще нет.

Операции Найти и Заменить

Поскольку способ представления информации в виде строки является универсальным, любую алгоритмическую задачу можно рассматривать как задачу обработки строковых данных. Оказывается, что можно так определить систему команд исполнителя, чтобы он, с одной стороны, обрабатывал только строки, а с другой - был способен выполнить любой алгоритм. Такую систему ко-

манд нашел в 50-х годах XX века А.А. Марков. Открытая им алгоритмическая система называется нормальными алгоритмами Маркова.

Основная операция нормального алгоритма Маркова - операция подстановки:

В слове W найти слово P и заменить его словом Q.

Эту операцию принято записывать формулой P => Q. Операция подстановки находит в строке первое слева вхождение слова P и заменяет его на слово Q. Марков вводит также заключительную подстановку P => Q. После применения заключительной подстановки алгоритм заканчивает свою работу.

Нормальный алгоритм Маркова является набором подстановок (обычных и заключительных):

p_1 è q_1, p_2 è q_2 , . . . , p_k è q_k

Нормальные алгоритмы Маркова используются в теоретических исследованиях алгоритмических проблем.

Форма записи строк

Чтобы отличить в тексте алгоритма строку от других примитивов (имени величины, числа и т.п.) строку принято брать в кавычки. Например: 3.14 - запись числа, а '3 .14 '- запись строки.

Операции над строками

В современных алгоритмических языках набор команд исполнителя (перечень операций над строками) по сравнению с алгорит-

мической системой Маркова значительно шире. В него могут входить вместе с командами сравнения, например такие команды:

Объединить строки P и Q.

Удалить из строки P, начиная с n-той буквы, строку из m букв.

Вставить в строку P, начиная с n-той буквы, строку Q.

Скопировать из строки P, начиная с n-той буквы, строку из m букв.

Определить длину строки P.

Определить в строке P наименьший номер буквы, начиная с которой в нее входит строка Q.

В алгоритмических языках для обозначе-

ния таких команд используются сокращения. Например:

Объединить (P, Q).

Удалить (P, n, m).

Вставить (P, n, Q).

Копировать (P, n, m).

Длина (P).

Номер (P, Q).

Для того, чтобы программист правильно использовал эти команды, он должен знать их форму записи и типы их аргументов и результатов. Например, для команды Копировать (P, n, m) имеем такую форму записи:

Q = Копировать (P, n, m).

Вход: строка P, натуральное n, натуральное m.

Выход: строка Q.

Эта команда имеет функциональную форму записи y = f(x). Поэтому команды такой формы называют функциями. Заметим, что результат этой команды программист должен "вернуть" в некоторую переменную. В нашем примере это переменная Q.

Команда Вставить (P, n, Q) имеет процедурную форму записи:

Вставить (P, n, Q).

Вход: строка P, натуральное n, строка Q.

Выход: строка P.

Заметим, что в этой команде результат совпадает с одним из аргументов: процедура Вставити (P, n, Q) изменяет свой первый аргумент - переменную P. Команды такой формы называют процедурами.

Таким образом, набор команд обработки строковых величин состоит из следующих процедур и функций:

Функция Объединить (P: строка, Q: строка) : строка.

Процедура Удалить (P: строка, n:натуральное, m:натуральное).

Процедура Вставить (P: строка, n:натуральное, Q: строка).

Функция Копировать (P: строка, n:нату-

ральное, m:натуральное): строка

Функция Длина (P: строка):натуральное.

Функция Номер (P: строка, Q: строка):натуральное.

1.9. Форма записи алгоритмов

Пример 1.5 Алгоритм решения приведенного квадратного уравнения $x^2 + px + q = 0$;

Алгоритм Приведенное квадратное уравнение **;**

Вход

p, q: Действительные числа;

Выход

Ответ: Буквенная величина;

x1, x2: Действительные числа;

Дополнительная величина

D: Действительное число;

Начало

Вычислить $D = p^2 - 4q$;

Если $D < 0$ **то** (Ответ **присвоить значение** ‘Решений нет’; **Перейти к** 1);

Если $D = 0$ **то** (**вычислить** $x1 = -p/2$; $x2$ **присвоить значение** x1; Ответ **присвоить значение** ‘Единственное решение’; **Перейти к** 1);

Ответ **присвоить значение** ‘Два реше-

ния’;

Вычислить x1 = (-p+Ö(D))/2;

Вычислить x2 = (-p-Ö(D))/2;

1:Конец.

Записывая этот алгоритм, мы старались сделать его максимально понятным читателю. Для этого мы:

· применили правила структурирования текста алгоритма;

· для однотипных команд использовали ту же форму записи;

· команды алгоритма записали в виде предложений украинского языка;

· команду Вычислить использовали для вычисления значения произвольного арифметического выражения, а не только для выполнения одной арифметической операции.

Правила такого типа определяют единую, стандартную для всех алгоритмов форму записи.

Обратите внимание на одну особенность записи алгоритмов: поскольку все команды записаны в виде предложений, текст алгоритма получается слишком многословным. Для того, чтобы устранить этот недостаток, при записи алгоритмов принято вводить сокращения, точно указывая их содержание.

Рассмотрим команду **Вычислить** $D = p^2 - 4q$**.** Общее правило ее записи можно сформулировать следующим образом:

Вычислить <Имя величины> = <Выражение>

Правило ее выполнения: исполнитель вычисляет значение выражения - правой части равенства и результат обозначает именем величины - левой части равенства. В программировании эту команду называют оператором (командой) присвоения. Во многих языках программирования для оператора присваивания принятое обозначение

< Имя величины >:= < Выражение >

Рассмотрим теперь команду

Ответ **присвоить значение** “Решений нет”

Правило ее записи также можно описать командой присвоения:

< Имя величины >:= <Данное>

Для команды

x2 присвоить значение x1

правило ее записи можно сформулировать так:

< Имя величины >:= < Имя величины >

Таким образом, обе эти команды – частные случаи команды присвоения.

Еще одно правило сокращения записи алгоритмов: для обозначения имен величин, типов, алгоритмов и т.д. используется одно служебное слово. Мы используем это правило, обозначив:

Приведенное квадратное уравнение – КвУравнение

Действительные_числа - ДЕЙСТ

Строковые_величины - СТРОКА

Некоторые слова в записи алгоритма играют служебную роль: они обозначают начало или окончание фрагмента, которые имеют свой особый смысл, отделяют один фрагмент от другого. Такие слова называют разделителями. В тексте алгоритма служебные слова-разделители выделены.

В нашем примере это слова Алгоритм, Вход, Выход, Вспомогательная величина, Начало, Конец, роль которых нам уже известна, а также другие слова, роль которых мы определим позднее.

Разделитель вспомогательная величина мы сократим до Доп.

Запись алгоритмов часто содержит математические обозначения. Надо сказать, что в традиционных математических обозначениях много особенностей, обусловленных историческими традициями. В алгоритмических языках, напротив, принята очень простая система математических обозначений. В этой системе обозначений все функции имеют имена. Вместо Ö(x) мы будем писать Sqrt(x).

Алгоритм КвУравнение;

Вход

p, q: ДЕЙСТ;

Выход

Ответ: СТРОКА;

x1, x2: ДЕЙСТ;

Доп

D: ДЕЙСТ;

Начало

D := p^2 - 4q;

Если D < 0

то (Ответ := “Решений нет”; Перейти 1**);**

Если D = 0

то (x1 := -p/2; x2 := x1;

Ответ := “Единственное решение”;

Перейти 1**);**

Ответ **:=** “Два решения”;

x1 **:=** (-p + Sqrt(D))/2;

x2 **:=** (-p - Sqrt(D))/2;

1:Конец.

Текст алгоритма стал компактнее. Однако, теперь для того, чтобы прочитать и понять, от читателя требуется специальная подготовка: он должен знать правила записи и интерпретации (толкования) и привыкнуть к ним.

Рассмотрим теперь те команды, которые мы использовали впервые. Во-первых, это команда

Если D < 0

то (Ответ := “Решений нет”; Перейти 1);

Эта команда имеет следующую общую форму записи:

Если <условие>

то (<последовательность команд>)

Выполняя эту команду, Исполнитель проверяет, выполнено ли условие, указанное после служебного слова **Если**. Если условие выполнено, Исполнитель переходит к выполнению последовательности команд, стоящих после слова **то** и заключенное в скобки. Если же условие не выполнено, Исполнитель переходит к выполнению следующей команды. Сравните эту команду с командой ветвления, которую мы использовали в алгоритме Евклида (пример 1.3) и Вы увидите, что она яв-

ляется упрощенным вариантом разветвления - неполным ветвлением.

Команды ветвления позволяют направить выполнения алгоритма по тому или иному пути в зависимости от выполнения условия. Такие команды называют командами выбора.

Команды выбора содержат в себе другие команды, которые выполняются в зависимости от результатов проверки условий.

При записи команды ветвления обычно используют правило структурирования, что представлено выше, или похожее правило

Если <условие> **то**

(<последовательность команд>)

Полное разветвления структурируют

обычно так:

Если <условие> **то**

(<последовательность команд >)

иначе

(<последовательность команд >)

Второй характерной командой в примере есть команда перехода Перейти 1. Она имеет форму записи

Перейти <N>,

причем число N используется в записи алгоритма как специальная метка той команды, которую нужно выполнить следующей.

В нашем примере используются команды перехода Перейти 1, а меткой 1 обозначена команда 1: Конец.

Выполнение команды перехода заключается в том, что Исполнитель переходит к выполнению команды, обозначенной меткой N (нарушая при этом естественную последовательность выполнения команд).

1.10. Команды управления

Команды выбора, повторения и перехода непосредственно не указывают на преобразование данных, а лишь на управление последовательностью этих преобразований. Поэтому их называют командами управления. Очевидно, что команды управления должны входить в систему команд любого исполнителя, претендующего на универсальность в некоторой предметной области.

В рамках рассматриваемого нами императивного подхода к описанию алгоритмов используют следующую классификацию команд управления: команды выбора, повторения, перехода. Мы уже ввели в употребление две команды выбора - команды полного и неполного ветвления, одну команду повторения и команду (безусловного) перехода. Некоторые другие команды управления будут рассмотрены позже. Еще раз обратим Ваше внимание на два важных аспекта в определении команды: ее форму записи (синтаксис) и ее содержание (семантику).

Синтаксис

Поскольку алгоритмы должны быть определены точно и недвусмысленно, форма их записи должна быть определена с математической точностью. С этой целью используются так называемые синтаксические правила. Совокупность синтаксических правил описа-

ния алгоритмов определяет формально-языковую среду - алгоритмический язык.

Совокупность синтаксических правил описаний алгоритма можно разбить на три группы правил:

· Общие правила оформления алгоритмов;

· правила описание величин;

· правила описания команд.

В рассмотренных примерах мы использовали следующие правила:

Правила оформления алгоритмов:

Алгоритм <Имя>;

Вход

<описание входных величин>;

Выход

<описание выходных величин>;

Доп

<описание дополнительных величин>;

Начало

<описание команд>

Конец.

Правила описание величин

Описание одной величины:

<Имя величины> **:** <Тип величины>

Описание нескольких величин одного типа:

<Имена величин через запятую> : <Тип величин>

Несколько описаний величин:

<Описание величин через точку с запятой>

Правила записи команд

Команда присвоения:

<Имя величины> := <Выражение>

Команды выбора:

Если <Условие> то <Команда>

Если <Условие> то <Команда> иначе <Команда>

Команда повторения:

Пока <Условие> выполнять <Команда>

Составная команда:

(<последовательность команд через точку с запятой>)

Составные команды состоят из других команд и рассматриваются исполнителем как одна команда. Правило предлагает брать несколько команд, отделенных друг от друга точкой с запятой, в круглые скобки. Мы использовали составные команды в примере 1.5 для того, чтобы объединить несколько ко-

манд в одну.

(x1 **:=** -p/2;

x2 := x1;

Ответ **:=** “Единственное решение”;

Перейти 1**)**

Команда перехода:

Перейти <Натуральное число>

Семантика

Совокупность правил, определяющих выполнение алгоритма, можно разбить на две такие группы:

· правила итерации данных;

· правила итерации команд.

Мы обратим сейчас внимание на правила интерпретации команд управления. Для определения этих правил мы будем использовать так называемые блок-схемы алгоритмов.

1.11. Блок - схемы

В старые добрые времена (50-60 годы XX века), когда компьютеры были большими и медленными, а языки программирования только начинали формироваться, для представления алгоритмов применяли блок-схемы, а для реализации алгоритмов в виде компьютерных программ - систему команд компьютера (машинный код).

Это делалось потому, что компьютерная

программа в машинных кодах совсем не приспособлена для ее чтения. Мы уже понимаем, что алгоритм нужно сначала изобрести, а только потом записать в форме, понятной исполнителю, т.е. в виде машинного кода. В процессе изобретения алгоритма программисты пользовались блок-схемами.

Блок-схема алгоритма - это его графическое изображение в виде нескольких блоков, соединенных между собой стрелками. Блоки изображают команды алгоритма, а стрелки - последовательность выполнения этих команд.

Команды, описывающие вычисления (например, команда присвоения) изображаются в виде прямоугольника

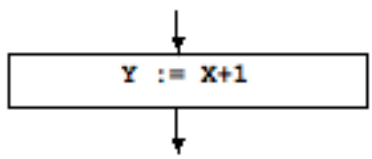


Рис 1.3. Блок Вычисления

Команда проверки условия выглядит как ромб:

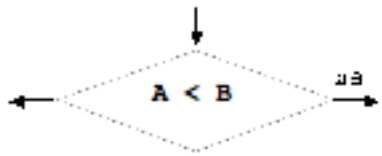


Рис 1.4. Блок Условие.

Начало и конец алгоритма изображают в виде овалов:

Рис 1.5. Блоки Начало и Конец.

Команды ввода и вывода данных мы изобразим так:

Рис 1.6. Блоки Ввода і Вывода.

Существуют и другие типы блоков, но мы ими пользоваться не будем.

1.12. Дополнительные алгоритмы

Изобразим сейчас блок-схема алгоритма 1.1, а рядом - блок-схема алгоритма Евклида 1.3.

Блоки, расположенные между блок-схемами алгоритма примера 1.1 и алгоритма Евклида, изображающие "настройки" алгоритма Евклида на работу с величинами алгоритма добавления дробей. Алгоритм добавления

дробей играет роль основного алгоритма, а алгоритм Евклида - вспомогательного.

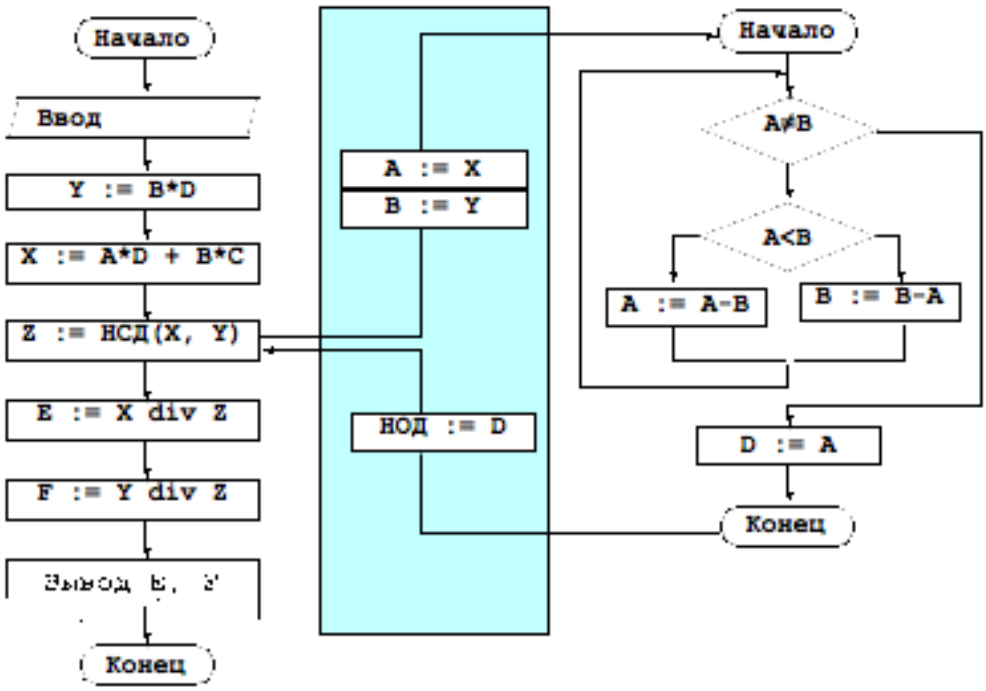

Начало
Ввод
Y := B*D
X := A*D + B*C
Z := НСД(X, Y)
E := X div Z
F := Y div Z
Вывод E, F
Конец
A := X
B := Y
НОД := D
Начало
A≠B
A<B
A := A-B
B := B-A
D := A
Конец

Рис 1.7 Основной алгоритм сложения дробей и вспомогательный алгоритм Евклида

Входные величины вспомогательного алгоритма перед тем, как он начнет выполняться, должны принять правильные значения. После того, как вспомогательный алгоритм вычислит значения исходных величин, эти величины должны быть переданы (возвращены) основному алгоритму.

Входные и выходные величины вспомогательного алгоритма называют его параметрами. Формирование значений параметров в

процессе взаимодействия основного и вспомогательного алгоритмов называют передачей параметров.

Передачу параметров во вспомогательный алгоритм берет на себя Исполнитель. Дело программиста - правильно указать фактическое значения этих параметров в основном алгоритме.

Мы показываем блок-схеме содержание (т.е. семантику) процесса взаимодействия основного и вспомогательного алгоритмов. В этом и заключается главное преимущество блок-схем: они позволяют представить процесс выполнения алгоритма в геометрических образах, существенно облегчая тем самым понимание семантики алгоритма.

Алгоритм сложения дробей выполняет команды последовательно. Если сравнить текст этого алгоритма с его блок-схемой, можно

убедиться в том, что особых преимуществ в наглядности изображения нет.

Поясним теперь на блок-схемах семантику команд ветвления и повторения.

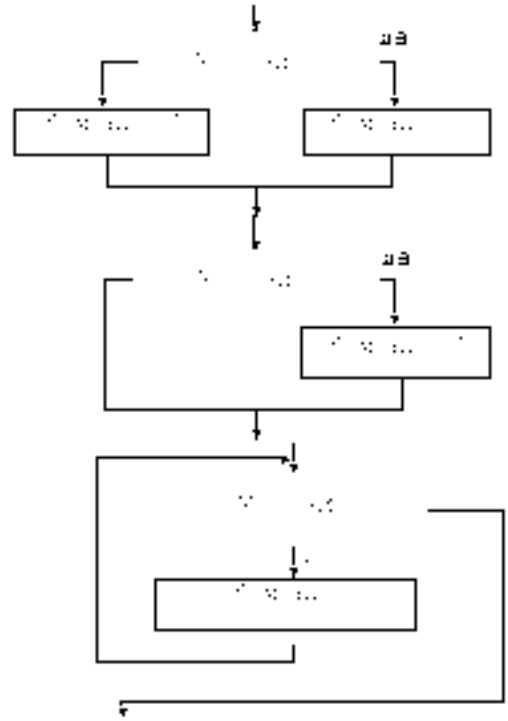
Команда	Блок-схема
Если <Условие> **То** <Команда 1> **Иначе** <Команда 2>	
Если < Условие > **То** <Команда 1>	
Пока < Условие > **выполнить** <Команда>	

Рис 1.8 Семантика команд управления

Блок-схема команды объясняет ее выполнения лучше, чем любое словесное описание.

Представьте себе, однако, что Вам предлагают рассмотреть алгоритм, блок-схема которого изображена на рис 1.9.

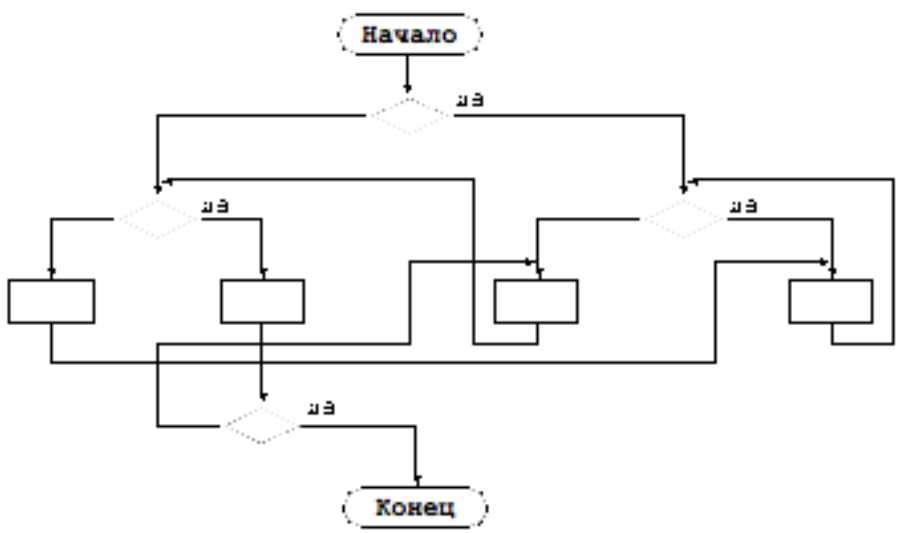


Рис 1.9. Блок-схема алгоритма типа «спагетти».

Автор алгоритма при этом утверждает, что его алгоритм работает правильно, т.е. всегда решает поставленную задачу.

Легко Вам разобраться в лабиринте переходов между блоками, и проверить логику вычислений? Конечно, нет. Каждая последовательность действий здесь так вплетена в другие возможные последовательности, что все это вместе взятое напоминает тщательно перемешанные спагетти.

С технической точки зрения причина образования спагетти заключается в том, что

блок Условие имеет два выхода, которые, в принципе, можно направлять куда угодно. Более важно то, что у нашего гипотетического автора не все в порядке с логикой мышления. Он не сумел выделить главные и вспомогательные способы управления вычислениями, не сумел структурировать алгоритм. Наличие же блока Условие с его двумя выходами позволяет некоторым образом скомбинировать переходы вместо того, чтобы направлять процесс построения алгоритма по пути определения логически обоснованной структуры управления.

Рассмотрим алгоритм решения следующей задачи:

Пример 1.6. Алгоритм приближенного решения уравнения f (x) = 0 методом деления отрезка пополам.

Дано:

Числовой отрезок [a, b].

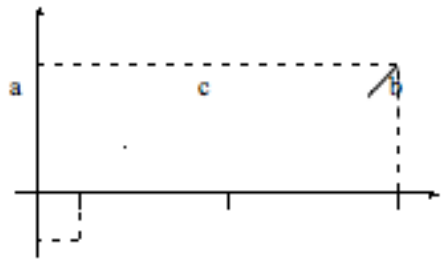
Функция f(x), непрерывна на отрезке и имеет на этом отрезке единственный корень xo.

t – заданная точность.

Найти:

Корень xo

В учебниках по вычислительной математике этот метод рассматривается как один из наиболее простых методов приближенного решения уравнений.



Метод:

Отрезок [a, b] разделим пополам точкой c. Определим, на которой из половин отрезка - [a, c] или [c, b] находится искомый корень. (Поскольку функция f (x) непрерывна и корень xo единственный, на концах отрезка, содержащего xo, функция f (x) принимает про-

тивоположные по знаку значения.) Деление отрезка пополам будем продолжать до тех пор, пока его длина не станет меньше, чем заданная точность t. Тогда приближенным значением xo можно выбрать среднюю точку этого отрезка.

Разрабатывая алгоритм, сначала определим его вход и выход:

Алгоритм Уравнение;

Вход

a, b, t: ДЕЙСТ;

Выход

Хо: ДЕЙСТ;

Начало

< последовательность команд алгоритма >

Конец.

Мы уточнили имена входных и выходных величин и определили их типы. Поставим перед собой вопрос: какой способ управления вычислениями является главным? Оказывается, что это - последовательное выполнение двух действий:

Начало

<Уменьшить длину отрезка к величине, меньшей t>;

<Вычислить x0 как середину отрезка [a, b]>

Конец.

Второе действие - это команда присвоения x0: = (a + b) / 2. Как можно уточнить первое действие? Это действие - повторение процедуры деления отрезка и выбора одной из его половин. Этот способ описывается командой повторения Пока <Условие> выполнять <Команда>. Поэтому нужно определить Условие и Команду.

Алгоритм Уравнение;

Вход

a, b, t: ДЕЙСТ;

Выход

Xo: ДЕЙСТ;

Начало

Пока <длина отрезка [a, b] больше либо равно t> **выполнить**

(

<Найти средину с отрезка [a, b];>

<Выбрать в качестве отрезка [a, b] ту половину,

на которой расположен корень>

);

x0 := (a+b)/2

Конец.

Уточняем Условие и команду вычисления середины отрезка. Вводим вспомогательную величину c:

Алгоритм Уравнение;

Вход

a, b, t: ДЕЙСТ;

Выход

Хо: ДЕЙСТ;

Доп

С: ДЕЙСТ;

Начало

Пока b –a >= t **выполнить**

(

c **:=** (a+b)/2;

<Выбирать ту половину, на которой расположен корень>

);

xo := (a+b)/2

Конец.

Уточняем действие выбора нужной половины отрезка:

Алгоритм Уравнение;

Вход

a, b, t: ДЕЙСТ;

Выход

Хо: ДЕЙСТ;

Доп

С: ДЕЙСТ;

Начало

Пока b –a >= t **выполнить**

(

c **:=** (a+b)/2;

Если <f(a) и f(c) принимает противоположные значение по знаку>

то b := c

иначе a := c

);

x0 := (a+b)/2

Конец.

Осталось уточнить Условие:

Алгоритм Уравнение;

Вход

a, b, t: ДЕЙСТ;

Выход

Хо: ДЕЙСТ;

Доп

C: ДЕЙСТ;

Начало

Пока b –a >= t **выполнить**

(

c **:=** (a+b)/2;

(

c **:=** (a+b)/2;

Если f(a)*f(c)< 0

то b := c

иначе a := c

);

x0 := (a+b)/2

Конец.

Разрабатывая этот алгоритм, мы опирались на его неформальное описание, изложенное в виде метода решения задачи. Программирование таких методов является главной работой программиста.

После того, как метод найден и признан подходящим, программист должен описать его в системе команд исполнителя, т.е. закодировать. Методика, которую мы показали, называется методом последовательных уточнений.

Действительно, мы начали с ответа на такой вопрос: что дано и что нужно найти. Уточнение входа и выхода в виде описаний величин - один из важнейших этапов разработки алгоритмов.

Далее, на каждом шаге построения алгоритма мы уточняли одно из действий метода решения задачи как команду или условие. Такая дисциплина позволяет ограничить рассуждения ответами на следующие вопросы:

Какой способ управления нужно сейчас применить?

Можно ли реализовать действие одной командой исполнителя?

Как сформулировать условие?

Программист, осознанно применяет ме-

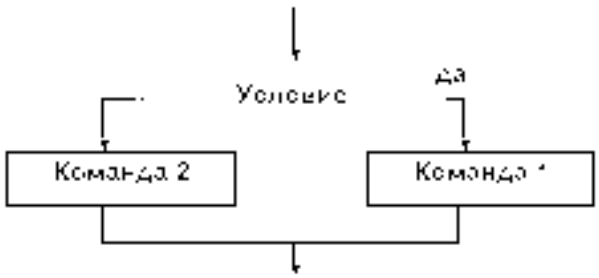
тод последовательных уточнений, экономит свое время, радует начальство качеством и скоростью работы, допускает мало ошибок и создает программы, которые легко читать другим программистам.

1.13. Базовые структуры управления

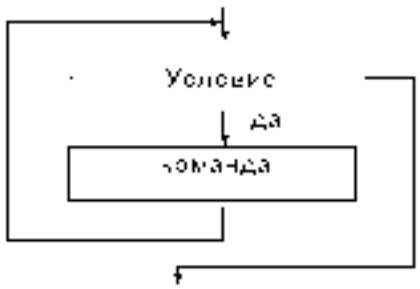
Основным достижением теории программирования 60-х годов является осознание и теоретическое осмысление того факта, что существуют несколько основных (базовых) способов управления вычислениями, используя которые можно описать любой алгоритмический метод решения задачи. Структура управления любого алгоритма может быть реализована в виде комбинации основных структур управления. Эти способы управления или структуры управления, нам уже известны. К ним относятся:

- **последовательное выполнение**

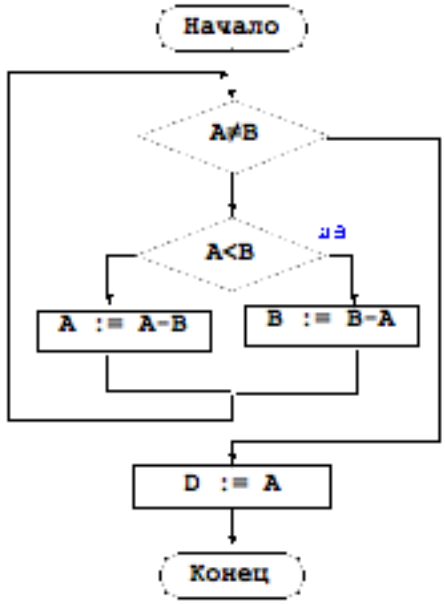
· **ветвление**



- **повторение с предусловием**



Таким образом, вместо программирования в системе команд - блоков Вычисление, Условие и Стрелка Перейти, можно программировать в структурах управления, причем быстродействие алгоритмов при этом не ухудшается, а такие свойства, как понятность («читабельность») алгоритма существенно улучшаются. Эффект спагетти не может проявиться тем, что каждая из структур управления имеет только один выход.



Алгоритм Евклида;

Вход A, B;

Выход: D;

{Комментарий: **D = НОД (AB)}**

A, B: НАТ;

D: НАТ;

Начало

Пока A ¹ B выполнить

Если A < B

то B := B - A

иначе A := A - B;

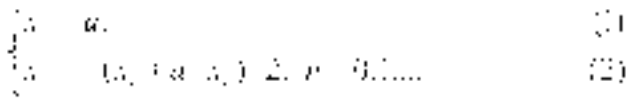
D := A

Конец.

Рис 1.10. Текст и блок-схема алгоритма Евклида.

Сравните текст и блок схему алгоритма Евклида (рис 1.9): структурированный текст представляет алгоритм не менее отчетливо, чем его блок-схема. Текст к тому же содержит информацию, которую нельзя выразить в графических образах. Это относится, например, к описанию величин.

Рассмотрим алгоритм приближенного вычисления квадратного корня из неотъемлемого действительного числа. Этот алгоритм опирается на метод приближенных вычислений, представленный формулами



где a – положительное число, с которого нужно вычислить корень.

Эти формулы нужно понимать так:

Вычисления начинаются со значения x = a (формула (1)).

На каждом шагу текущее значение x подставляется в правую часть формулы (2) и вычисляется новое значение x.

Если новое значение x отличается от старого на величину большую, чем точность вы-

числений, вычисления по формуле (2) повторяется. В противном случае вычисления закончено.

В формулах метода число n играет роль счетчика числа шагов .

Запишем этот алгоритм:

Алгоритм КвКорень;

Вход a, t: ДЕЙСТ;

Выход y: ДЕЙСТ;

Доп x: ДЕЙСТ;

Начало

y := a;

Повторять

x := y;

y := (x + a/x)/2

до Abs(x – y)<= t

Конец.

Методами математического анализа можно доказать, что значение y может быть сделано как угодно близким к

1.14. Абстракция данных

Примеры исполнителей, рассмотренные выше, используют абстрактные представления о предметных областях, над которыми определяются алгоритмы. Они показывают,

что понятие Исполнителя является одной из основных абстракций, используемых для описания алгоритмов. Именно система команд Исполнителя является уточнением набора средств, с помощью которых строятся алгоритмы. В наших примерах системы команд Исполнителя предметно-ориентированные. Так, в примере 1.1 Исполнитель имеет дело с целочисленной арифметикой, а в задаче 1.4 - с арифметикой действительных чисел. Особенно наглядно это демонстрирует Исполнитель примера 1.2, команды которого выполняют геометрические построения.

Разработчик конкретного прикладного алгоритма (программист) или проецирует его в терминах ранее определенного набора команд Исполнителя или предварительно описывает (проектирует) систему команд Исполнителя, создавая тем самым понятный аппарат и инструментарий предметной области Исполнителя, а затем использует его в описа-

нии прикладного алгоритма.

Проблема проектирования набора команд Исполнителя как описания свойств и методов соответствующей предметной области играет центральную роль в теории и практике алгоритмизации. Именно ей посвящены практически все научные исследования и учебные курсы по программированию.

Совокупности данных, которые обрабатывает алгоритм, принято называть структурами данных (Исполнителя). Структуры данных (способы описания упорядоченных данных) влияют на эффективность алгоритма, а также на простоту его понимания и программной реализации. Структуры данных являются главными строительными блоками, из которых формируются алгоритмы. Алгоритмические структуры данных должны, с одной стороны, адекватно описывать ту предметную область, над которой определяется

Исполнитель, с другой - эффективно отображаться в универсальные структуры данных (например, в структуру компьютерной памяти). Эти в какой-то степени противоречивые требования приводят к следующему, более формальному пониманию термина «структура данных».

Структура данных состоит из трех основных компонентов:

· Набор предметно-ориентированных операций для обработки специфических типов абстрактных объектов описываемой предметной области.

· Структура памяти, в которой хранятся данные, описывающие абстрактные объекты.

· Интерпретация (реализация) каждой из операций в терминах структуры памяти.

Первый компонент определения - набор операций над абстрактными объектами - называется абстрактным типом данных (АТД). Второй и третий компоненты вместе образуют реализацию структуры данных.

АТД определяет, что делает структура данных - какие операции она поддерживает, не раскрывая, как они выполняются.

Пример 1.7. АТД Планиметрия (исполнитель Геометр)

Примитивные типы объектов: точка, *прямая, круг.*

Примитивные операции:

Прямая: (*точка, точка*) à *прямая*

l = Прямая(A, B) определяет прямую **l**, которая проходит через точки **A, B**

Круг: (*точка, точка*) à *круг*

o = Круг(A, B) определяет круг **o** с центром в точке **A,** которая проходит через точку **B.**

Круг: (*точка, точка, точка*) à *круг*

o = Круг(A, B, C) определяет круг **o** с центром в т. **A,** построено раствором циркуля с ножками, вставленные в **B**, **C**.

Точка (*прямая, прямая*) à *точка*

A = Точка(l, m) определяет *точку* **A** пересечения прямых **l** и **m**

ПересечениеКругов(*круг, круг*) à (*точка, точка*)

ПересечениеКругов (o, p, A, B) определяет *точки* **A**, **B** пересечения кругов **o** и **p**.

ПересечениеПрямаяКруг(*прямая, круг*) à (*точка, точка*)

ПересечениеПрямаяКруг (l, o, A, B) определяет точки **A**, **B** пересечения прямой **l** и круга **o**.

Из примитивных типов объектов АТД Планиметрия можно теперь строить составные (производные) типы. Например тип треугольник можно определить как тройку точек (вершин)

Треугольник = (точка, точка, точка)

Из примитивных (основных) операций АТД можно определять производные операции. Например, операция

ParallelLine: (точка, прямая) à прямая,

результатом, которой является прямая, проходящая через данную точку и параллельна данной прямой, может быть определена через примитивные операции и использована затем в алгоритмах решения задач на построения. Отметим, что реализация (интерпретация) АТД можно осуществлять отдельно, используя структуры данных, ориентированные на структуру памяти компьютера.

Использование АТД приводит к применению модульного подхода к алгоритмизации

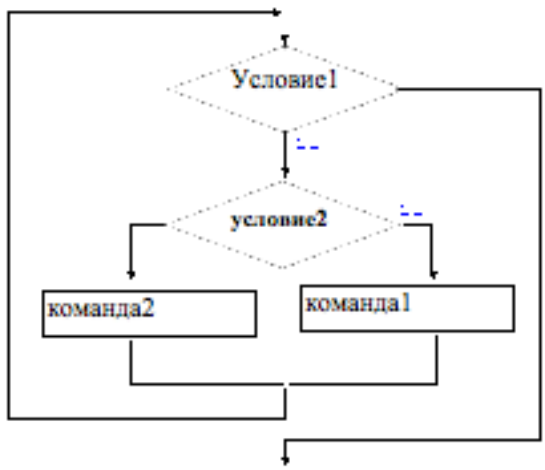
(программирования), то есть к практике разбивки алгоритма на отдельные модули (библиотеки) с хорошо отработанным интерфейсом.

1.15. Структурное программирование

Описание алгоритмов в терминах структур данных и структур управления называют структурным программированием. Особую роль в структурном программировании играют процедуры и функции - основные семантические единицы проектируемого алгоритма, содержащие описания отдельных подзадач, имеющие самостоятельное значение. Процедуры содержат описания интерпретаций (реализаций) АТД. Поэтому структурный подход называют также процедурным программированием.

Программирование управление осуществляется комбинированием основных структур

управления. Например, комбинирование ветвления и повторения приводит к блок-схеме



В реальных языках структурного програм-

мирования применяют и другие управляющие структуры, каждую из которых можно отнести к одному из трех основных типов. Например, в языке Pascal операторы выбора - условный оператор, короткий условный оператор, оператор варианта; операторы повторения - оператор цикла с параметром, оператор цикла с предусловием, оператор цикла с постусловием. Наличие дополнительных управляющих структур порождает избыточность в выразительных возможностях языка, что позволяет делать программы более естественными.

Отметим, что оператор перехода Перейти_ <N> (оператор Goto) не включен ни в список основных, ни дополнительных операторов управления. Это - не случайность. Как мы видели, бесконтрольное применение этого оператора приводит к тому, что алгоритм теряет свойства, указанные выше. Поэтому структурное программирование иногда назы-

вают программированием без Goto.

Проблема проектирования набора команд Исполнителя как описания свойств и методов соответствующей предметной области играет центральную роль в теории и практике алгоритмизации. Именно ей посвящены практически все научные исследования и учебные курсы по программированию.

1.16. Парадигма процедурного программирования

Основная идея, методология (парадигма) структурного подхода к определению алгоритма может быть выражена «формуле» Н.Вирта:

АТД + Структуры управления = Алгоритмы

Алгоритмы + Структуры данных = Программы.

Несмотря на все разнообразие форм представления информации и операций ее преобразования, которые использует человек в своей деятельности, оказалось возможным создание универсального Исполнителя, система команд которого позволяет промоделировать любую другую систему команд. Таким Исполнителем является компьютер.