НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

В. Н. Суриков, А. С. Кудрявцев, Г. А. Петров
Е.В. Хардиков

ОСНОВЫ АЛГОРИТМИЗАЦИИ ИНЖЕНЕРНЫХ ЗАДАЧ

УЧЕБНОЕ ПОСОБИЕ

Санкт-Петербург
2012

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ»

В.Н. Суриков, А.С. Кудрявцев, Г.А. Петров
Е.В. Хардиков

ОСНОВЫ АЛГОРИТМИЗАЦИИ ИНЖЕНЕРНЫХ ЗАДАЧ

УЧЕБНОЕ ПОСОБИЕ

2-е издание

Рекомендовано УМО вузов РФ по образованию в области радиотехники, электроники, биомедицинской техники и автоматизации в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 220201 "Управление и информатика в технических системах"

Санкт-Петербург
2012

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ББК 31.38я7
О-752
УДК 681.3.6(075)

Суриков В.Н., Кудрявцев А.С., Петров Г.А., Хардиков Е.В.

Основы алгоритмизации инженерных задач: учеб. пособие.-2-е изд.-СПБ. СПбГТУ РП. СПб.,2012. - 158 с.: ил.49,- ISBN 978-5-91646-050-6

В пособии изложены теоретические вопросы алгоритмизации применительно, к решению инженерных задач на ЭВМ, рассматриваются алгоритмы управления непрерывными и дискретными процессами в АСУТП, вопросы организации и использования различных структур данных и файлов, языки программирования контроллеров АСУТП и технология разработки задач для решения на ЭВМ, дается характеристика языков программирования высокого уровня.

Пособие предназначено для студентов, обучающихся по специальности “Управление и информатика в технических системах"’, при изучении дисциплин “Программирование и основы алгоритмизации”, “Программирование и отладка контроллеров”, “Автоматизация производственных процессов”.

Рецензенты: профессор кафедры автоматики и автоматизации производственных процессов СПбГУН и ПТ, доктор технических наук, профессор В.А. Балюбаш;
зав. кафедрой автоматизации химико - технологических процессов СПбГТУ РП, доктор технических наук, профессор Г. А. Кондрашкова.

Рекомендовано к изданию Редакционно-издательским советом университета в качестве учебного пособия.

ISBN 978-5-91646-050-6 © Суриков В.Н., Кудрявцев.А.С., Петров Г. А., Хардиков Е. В., 2012
© Санкт-Петербургский государственный технологический университет растительных полимеров, 2012

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Учебное издание

Валерий Николаевич Суриков
Александр Сергеевич Кудрявцев
Геннадий Алексеевич Петров
Евгений Васильевич Хардиков

ОСНОВЫ АЛГОРИТМИЗАЦИИ ИНЖЕНЕРНЫХ ЗАДАЧ

Учебное пособие

Редактор и корректор Т.А. Смирнова
Технический редактор Л.Я. Титова Темплан 2012, поз. 112
Подп. к печати 25.12.12.Формат 60x84/16. Бумага тип.№1. Печать офсетная.
Печ. л. 10,0. Уч.-изд. л. 10,0. Тираж 150 экз. Изд. №112. Цена «С». Заказ

ризограф Санкт-Петербургского государственного
технологического университета растительных полимеров, 198095, СПб.,
Ул. Ивана Черных, 4.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Предисловие

В настоящее время невозможно представить инженера, не обращающегося в своей работе к ЭВМ. Диапазон их общения определяется наличием прикладных программных средств, позволяющих пользователю решать нужные инженерные и научные задачи. При этом он должен и уметь пользоваться ранее созданными программами и участвовать в разработке нового программного обеспечения. Пользователь-инженер, включенный в коллектив разработчиков задачи для ее решения на ЭВМ, а, тем более, выполняющий эту работу лично, должен уметь “общаться” с алгоритмами. Разработка новой задачи всегда начинается с оценки возможности использования уже созданных программ. При такой оценке необходимо понять суть алгоритма рассматриваемой программы. Если он полностью удовлетворяет пользователя, то можно остановиться на этой программе. Если же этот алгоритм отличается от предлагаемого разработчиками, то следует рассмотреть возможность доработки анализируемой программы. И только в тех случаях, когда такую доработку выполнить невозможно или для новой задачи не удается найти ранее созданных аналогов, возникает необходимость ее полной разработки, включая написание новой программы.

Как видно из изложенного, эффективность общения пользователя с ЭВМ в большой степени определяется его познаниями в вопросах алгоритмизации.

Алгоритмизация задач для решения на ЭВМ сформировалась к настоящему времени в самостоятельную научно-прикладную дисциплину, к которой обращаются специалисты различных областей деятельности. Основой дисциплины является утверждение, что понятие алгоритма является основным при составлении компьютерных программ.

В предлагаемом пособии излагаются вопросы алгоритмизации в объеме, достаточном для изучения технологии разработки подлежащих решению на ЭВМ инженерных задач, применительно к специальности обучаемых. Пособие состоит из шести глав.

В первой главе приводится определение и даются общие характеристики алгоритма, исходя из его современных приложений, рассматриваются основные свойства алгоритмов, учитываемые при их разработке, дается понятие алгоритмической системы и общая характеристика систем применительно к различным классам задач.

Во второй главе рассматриваются возможные способы описания алгоритмов, дается понятие базовых алгоритмических структур, излагаются правила описания линейных, разветвляющихся и циклических алгоритмов.

В третьей главе приведено большое число типовых алгоритмов, находящих применение при решении различных инженерных задач. Излагаемый материал проиллюстрирован большим количеством примеров.

В четвертой главе рассматриваются алгоритмы управления непрерывными и дискретными процессами, используемые в автоматизированных системах управления.

Пятая глава посвящена вопросам рассмотрения организации и использования различных структур данных и файлов. Структуры данных наряду с алгоритмами определяют структуру и основные характеристики разрабатываемых программ.

В шестой главе дается характеристика языков программирования высокого уровня, рассматриваются языки программирования контроллеров АСУТП, излагается содержание разработки алгоритмов и программ, разбираются понятия жизненного цикла, надежности и сопровождения программного обеспечения.

Пособие написано профессором Суриковым В.Н. (глава 3, общая редакция) и доцентами Кудрявцевым А.С.(введение, главы 1,2,3), Петровым Г.А.(главы 5,6), Хардиковым Е.В.(глава 4).

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Глава 1. Основные понятия алгоритмов

1.1. Определение и свойства алгоритма

Происхождение термина алгоритм и его исходное содержание принадлежат математике. Это слово происходит от Algorithmi - латинского написания имени Мухаммеда аль-Хорезми, выдающегося математика средневекового Востока, жившего и работавшего в XI веке в Хорезме. В XII веке был выполнен латинский перевод его математического трактата, из которого европейцы узнали о десятичной позиционной системе счисления и правилах выполнения в ней арифметических действий. Именно эти правила в то время называли алгоритмами. Сложение, вычитание, умножение столбиком, деление уголком многозначных чисел - вот первые алгоритмы в математике.

Определение алгоритма как процесса вычислений оказалось достаточным вплоть до середины XX века. К этому времени *под алгоритмом понимали конечную совокупность точно формулированных правил, которые позволяют решать те или иные классы задач.* Каждому оригинальному процессу вычислений присваивалось имя, иногда имя автора, например, алгоритм деления с остатком, алгоритм нахождения простых чисел, алгоритм Евклида.

Данное определение алгоритма не является формально определённым, гак как в нём не содержится того, что следует понимать под правилами решения задач. Однако в течение длительного времени математики довольствовались им, поскольку практически не возникало случаев расхождения во мнениях относительно того, является ли алгоритмом тот или иной конкретно заданный вычислительный процесс.

Оценка приведённого выше определения алгоритма существенно изменилась, когда в математике возникла необходимость обоснования алгоритмической разрешимости и неразрешимости задач. Оказалось, что одно дело доказать существование алгоритма решения некоторой задачи, другое - доказать его отсутствие. Первое можно сделать путём описания процесса ре-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

шения задачи. В этом случае достаточно приведённого выше определения. Доказать несуществование алгоритма таким путём невозможно. Для этого надо точно знать, что такое алгоритм.

Потребность решения приведённой и ряда других теоретических проблем привели к выделению в середине XX века теории алгоритмов в качестве самостоятельного научного направления. Исходным шагом новой дисциплины стала формализация определения алгоритма, а также характеристика его свойств. В 30-е годы XX века ряд математиков попытались уточнить понятие алгоритма, а затем, исходя из него, определить точно класс вычисляемых функций. Основная идея заключалась в том, что алгоритмические процессы может совершать специально построенная “машина” (устройство, автомат). В соответствии с этой идеей были описаны в точных математических терминах довольно узкие классы машин, но на этих машинах оказалось возможным осуществить (промоделировать) все алгоритмические процессы, которые когда-либо описывались математически. Машинами эти математические построения были названы потому, что при их описании использовались некоторые понятия реальных ЭВМ - память, команда, программа.

Существенным фактором, повлиявшим на развитие теории алгоритмов, явилось создание электронных вычислительных и управляющих машин. Крупнейший специалист по компьютерной математике Д.Кнут пишет: “Алгоритм должен быть определён настолько чётко, чтобы его указаниям мог следовать даже компьютер. Понятие алгоритма является основным при составлении компьютерных программ. Убыстряющееся развитие вычислительной техники и программирования будет включать в алгоритмическую практику всё большую часть разумной деятельности человека”

Уже на начальном этапе использования ЭВМ в качестве средства для автоматического производства вычислений (“автоматических арифмометров”) оказалось, что принятые в математике правила записи формул недостаточно

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

строго формальны для их “восприятия” компьютером. Например, ввод в ЭВМ понятной для человека формулы:

$$\omega = \frac{\alpha\beta}{\gamma}$$

приведёт не к выдаче результата вычислений, а к сообщениям об ошибках в её задании.

Различной оказалась и необходимая степень детализации вычислительных алгоритмов для человека и ЭВМ. Во втором случае алгоритм решения задачи должен быть представлен в виде описания, однозначно определяющего последовательность и содержание пошаговых действий компьютера, обеспечивающих получение решения, что не всегда обязательно для человека.

Проиллюстрируем изложенное примером. Пусть задача состоит в определении среднего арифметического nчисел.

При производстве вычислений вручную или с помощью калькулятора достаточно вспомнить формулу:

$$\mathrm{M}_n = \frac{a_1 + a_2 + \ldots + a_n}{n}$$

или сокращённо:

$$\mathrm{M}_n = \frac{\Sigma a_i}{n}$$

приi=1,2…n; $\sum a_i$=S,

а затем последовательно выполнить операции сложения и деления.

Для ЭВМ процесс решения задачи может быть записан при заданных значенияхnи $a_1,\ldots,a_n$ в виде следующей системы последовательных указаний, т.е. алгоритма:

1. Принять i=0 и перейти к следующему шагу.

2. Принять S=0 и перейти к следующему шагу.

3. Принять $i=i+1$и перейти к следующему шагу.

4. Принять $S=S+a_i$ и перейти к следующему шагу.

5. Проверить, выполняется ли условие $i=n$, и, если не выполняется, то вернуться к шагу 3; если выполняется, то перейти к следующему шагу.

$$М_n = \frac{S}{n}$$

6. Вычислить

Развитие компьютерной техники и накопленный опыт её применения привели в течение достаточно короткого промежутка времени (10-15 лет после появления первых ЭВМ) к существенному расширению содержания понятия алгоритма. Это определилось тем, что компьютеры, первоначально предназначенные для автоматического выполнения вычислений, оказались универсальным инструментом обработки данных самого различного происхождения и характера. Они оказались способными автоматически выполнять алгоритмы анализа и преобразования разнообразных символьных последовательностей, и этого оказалось достаточно, чтобы использовать их для решения задач, не имеющих отношения к расчётам: задачи типа анализа текстов, формирования чертежей, накопления, хранения, обновления информации.

Термин *алгоритм* в его современной трактовке находится в тесной связи с понятиями получения (выработки, нахождения, обоснования, принятия) решений и их выполнения. Поначалу математический, к настоящему времени он стал привычным не только для математиков, но и для специалистов, связанных с управленческой деятельностью в самых различных областях её приложения. В соответствии с этим, наряду с исходным термином, ныне ши-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

роко применяются термины *алгоритм планирования* и *алгоритм управления.* Последними определяется возможное использование алгоритмов в качестве средства выработки решений, а затем и средства руководства выполнения принятых решений. В частности, во многих современных технических системах, выработка решений (управляющих воздействий) и их последующая

реализация возложены на автоматы, функционирование которых определяется соответствующими алгоритмами.

Применение компьютеров и микропроцессорной техники в автоматических системах управления потребовало разработки управляющих алгоритмов, направляющих работу этих систем во всех возможных режимах, включая и аварийные. Рассмотрим в качестве примера укрупнённый алгоритм работы автоматической системы управления уровнем воды в резервуаре, который может быть задан приведённой ниже последовательностью действий.

1. Включение системы в рабочий режим.

2. Запрос системой задания значений исходных данных (исходными данными являются значения нижней и верхней границ уровня воды –L_1 и L_2). Ввод пользователем с пульта значений исходных данных.

3. Перевод пользователем системы в автоматический режим работы (начало цикла управления).

4. Опрос датчика (получение фактического уровня воды в резервуаре на момент опроса –$L_{изм}$).

5. Сравнение $L_{изм}$ с L_1, При $L_{изм} \leq L_1$ перейти к шагу 7. Иначе перейти к следующему шагу.

6. Сравнение $L_{изм}$ с L_2. Если $L_{изм} \geq L_2$, то перейти к следующему шагу. В противном случае перейти к шагу 8.

7. Включение в работу модуля выдачи управляющего воздействия (исходя из значения L_1 (L_2) определяется степень открытия (закрытия) задвижки).

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

8. Конец цикла управления (при использовании аналоговой аппаратуры управления цикл повторяется непрерывно, для цифровой - дискретно).

Применительно к современным областям приложения *алгоритм определяют как последовательность однозначно определённых команд (инструкций) исполнителю (человеку или автомату), выполнение которых в конце данной последовательности приводиткдостижению поставленной цели.*

Определение позволяет рассматривать любой алгоритм как описание некоторого управляющего процесса, обеспечивающего перевод системы из исходного состояния в конечное, соответствующее поставленной цели. Иначе можно сказать, что алгоритмом задаётся алгоритмический процесс, под которым понимается процесс последовательного преобразования состояния объекта, происходящий дискретными шагами, где каждый очередной шаг состоит в смене одного состояния другим. Примерами алгоритмических процессов являются, в частности, описание решения математической задачи, процесс выработки управляющих воздействий, а также задание порядка их реализации. Ниже приводятся общие характеристики, присущие любому алгоритмическому процессу.

1. Алгоритм, применённый ко всякому набору исходных данных из их допустимого множества (начальному состоянию системы) S_0, приводит к решению (заключительному состоянию системы) S_k.

2. Алгоритмический процесс расчленяется на отдельные шаги заранее ограниченной сложности. Каждый шаг состоит в непосредственной переработке, возникшего к этому шагу состояния S_i в состояние S_{i+1}.

3. Непосредственная переработка S_i в S_{i+1} производится лишь на основании информации о виде заранее определённого состояния S_i.

4. Процесс переработки S_o в S_1, S_1 в S_2 и т.д. продолжается до тех пор, пока не появится информация о получении решения (достижения состояния S_k), либо не произойдёт безрезультативная остановка при невозможности достижения заданной цели с соответствующим сообщением об этом.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

В современной жизни алгоритм - понятие часто встречающееся в различных областях деятельности людей. Современное его значение во многом аналогично таким понятиям, как рецепт, способ, метод, процедура, процесс, программа. Однако понятие алгоритма имеет дополнительный смысловой оттенок. Алгоритм - это не просто набор конечного числа инструкций, задающих последовательность выполнения операций для решения задачи определённого типа. Помимо этого любому алгоритму присущи пять специфических свойств, характерных для всех алгоритмов. Такими *общими свойствами алгоритмов являются: дискретность, массовость, результативность, понятность, однозначность.*

Первым свойством алгоритма является *дискретность,* т.е. пошаговый характер определяемого им процесса. Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность чётко разделённых друг от друга предписаний (директив, команд), образующих дискретную структуру алгоритма. Только выполнив требования одного предписания, можно приступить к выполнению следующего.

Другим важным свойством алгоритмов является *массовость.* Смысл данного понятия заключается в том, что для любого рассматриваемого алгоритма существует некоторое множество вариантов входных данных, а, следовательно, и множество возможных исходов.

Любой алгоритм должен обладать свойством *результативности* (иначе, *конечности).* Это свойство означает, что при точном исполнении всех его предписаний, а также какими бы ни были значения исходных данных, процесс должен прекратиться за конечное число шагов и при этом должен быть получен определённый результат решения задачи.

Свойство *понятности* алгоритма означает, что исполнитель (человек, компьютер, автоматическое устройство управления) понимает его предписания и в состоянии их выполнить. При этом он действует “механически”. Естественно, при составлении описания алгоритма можно использовать только

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

те предписания, которые приведут к получению нужного результата.

Однозначность (иначе, *определённость)* алгоритма означает, что одно и то же предписание, будучи понятным разным исполнителям, после выполнения каждым из них должно давать одинаковый результат.

Запись алгоритма должна быть настолько полной и продуманной в деталях, чтобы у исполнителя никогда не могло возникать потребности в приня-

тии каких-либо самостоятельных решений, не предусмотренных составителем алгоритма. Очевидно, что после выполнения очередного предписания алгоритма исполнителю должно быть однозначно ясно, какое из предписаний должно выполняться на следующем шаге.

Рассмотрим на примере необходимость обязательного учёта приведённых выше свойств при построении алгоритма. Составим алгоритм решения квадратного уравнения $ax^2+bx+c=0$

Задача хорошо знакома из математики. Для её решения применительно к возможным значениям исходных данных разработаны несколько алгоритмов. В общем случае решением будут значения двух корней x_1 и x_2, которые вычисляются но формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Запишем алгоритм решения задачи в виде следующей последовательности инструкций:

1. Задать значения *a, b, c.*
2. Вычислить значение дискриминанта:

$$d=b^2-4ac.$$

3. Вычислить значение первого корня:

$$x_1 = \frac{-b + \sqrt{d}}{2a}$$

4. Вычислить значение второго корня:

$$x_2 = \frac{-b - \sqrt{d}}{2a}$$

Несовершенство приведённого алгоритма очевидно. Он не обладает свойством результативности. Обучаемым предлагается самостоятельно обосновать причину такого вывода.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

1.2. Алгоритмические системы. Общие правила построения алгоритмов

Важным понятием, сформулированным и широко используемым в теории алгоритмов, является понятие алгоритмической системы. *В теории алгоритмов под алгоритмической системой принято понимать набор средств и понятий, позволяющих строить множество алгоритмов, дающих возможность получать решение различных задач.*

В практике построения алгоритмов используется множество разнообразных алгоритмических систем. Можно говорить, например, об алгоритмических системах вычислений, системах графических построений и системах работы с текстами. Сюда относятся также и различные системы, обеспечивающие разработку алгоритмов автоматического управления, например, алгоритмов управления роботами-манипуляторами. Потребность создания и применения различных алгоритмических систем определяется:

-возможностями исполнителей алгоритмов;

-языком, на котором формируются алгоритмы, адресованные конкретному исполнителю;

-классом решаемых задач;

-множеством и характеристиками входных объектов, подлежащих обработке алгоритмами данной системы;

-множеством и характеристиками выходных объектов, определяющих результаты выполнения алгоритмов в данной системе.

Из изложенного выше (п.1.1) видно, что возможности исполнителей в большой степени определяют содержание алгоритмов решения задач. Поэтому алгоритмы, обеспечивающие решение одной задачи, но ориентированные на различных исполнителей, могут существенно различаться между собой: так, например, различны наборы действий, которые могут выполнять человек и компьютер в процессе решения задач обработки данных. Если исполнитель-человек, то алгоритм выбора максимального числа из нескольких может состоять из одного шага (визуально сравнить и выбратъ). В то же время вычислительная машина не может

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

решить в одно действие такую простую задачу, и нахождение решения для нее превращается в несколько шагов алгоритма.

Следовательно, при построении алгоритма предполагаются известными действия, которые может выполнять его исполнитель. Или, иначе говоря, предполагаются известными возможности исполнителя алгоритма. Например, в задаче решения квадратного уравнения исполнитель должен уметь выполнять действия сложения, вычитания, умножения, деления и извлечения квадратного корня.

Язык алгоритмической системы должен быть точно понимаем исполнителем. Если исполнитель алгоритма — человек, владеющий русским языком, то в качестве языка описания создаваемого для него алгоритма может быть использован русский язык.

Язык описаний, на котором задаются шаги заданий, адресованных компьютеру, значительно беднее, чем язык, адресованный человеку, так как набор действий, которые может выполнять компьютер, сравнительно невелик. Однако, этот язык более формален, его предложения не допускают различных толкований, и алгоритм, написанный на таком языке, представляет собой последовательность команд, не похожую на привычные нам фразы естественного языка. То, что язык ЭВМ назван языком, определяется наличием в нём, подобно любому языку общения людей, своего алфавита и своей грамматики. Соответственно, можно считать, что алгоритм, адресованный ЭВМ как исполнителю, представляет собой некий текст. Всякий текст на любом языке, в том числе и тексты алгоритмов, есть некоторая символьная последовательность, которую можно преобразовать в той же алгоритмической системе. Следовательно, открывается возможность разрабатывать специальные алгоритмы, которые будут преобразовывать алгоритмы, обрабатывая тексты их описаний. При решении задач с помощью ЭВМ этим широко пользуются, автоматически преобразуя алгоритмы из одной формы в другую или автоматически меняя алгоритм в ходе вычислений. Это создает логическую гибкость, которая и превратила ЭВМ в

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

принципиально новый инструмент обработки данных. Недаром говорят об искусственном интеллекте, об интеллектуальных системах, построенных на основе ЭВМ, и недаром пытаются с помощью ЭВМ моделировать работу мозга человека.

Структура и возможности использования алгоритмической системы для описания алгоритмов определяется также классом задач, подлежащих решению. Система должна обеспечивать описание всего необходимого набора действий, выполнение которых может потребоваться в процессе решения задачи.

В различных алгоритмических системах действия исполнителя могут быть резко различны. Например, в простейших задачах, связанных с производством вычислений, исполнитель должен знать правила и очередность выполнения арифметических действий. В задачах геометрических построений предполагается, что исполнитель может выбирать произвольные точки, соединять их, проводить прямые линии, чертить окружности, находить точки пересечения и т.д. В алгоритмах управления роботами-манипуляторами, в зависимости от конструкции робота, предполагаются известными те элементарные действия, которые он может выполнять, например, поворот механической руки, ее движение вперед или назад, захват детали и т.д.

Алгоритмическая система должна предусматривать возможность проведения анализа значений исходных и выходных характеристик применительно к конкретной задаче с тем, чтобы обеспечивалось ее решение.

Когда идет речь о построении алгоритма решения какой-либо конкретной задачи, то предполагаются известными все те объекты, которые будут исходными данными для него. Применительно к рассмотренному в предыдущем параграфе примеру решения квадратного уравнения, такими исходными объектами являются триады любых чисел, определяющие значения коэффициентов уравнения, т.е. a, b, c. Анализ заданных значений коэффициентов должен определить обращение к тому или иному алгоритму.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Не всегда конкретный алгоритм, построенный в выбранной алгоритмической системе, будет пригоден для работы со всеми ее входными объектами. Так, например, в алгоритмической системе, предназначенной для описания алгоритмов обработки числовой информации, в качестве исходных данных могут быть определены все числа. Однако, в тех алгоритмах, которые включают в себя операцию деления, в качестве значения исходных данных, записываемых в знаменателе формулы, нельзя использовать ноль, хотя он и принадлежит к множеству данных этой алгоритмической системы. Тем самым, для конкретного алгоритма не все исходные данные оказываются допустимыми. В этих случаях говорят, что данный конкретный алгоритм не применим к таким исходным данным. В таких случаях в алгоритме или дается предварительное указание о невозможности использования тех или иных конкретных данных, или вводится дополнительный шаг, обеспечивающий специальную проверку задаваемого набора значений исходных данных.

Из свойств алгоритма следует, что при выполнении алгоритма результат должен быть получен за конкретное число шагов. Если же окажется, что при применении алгоритма к каким-то исходным данным алгоритмический процесс продолжается бесконечно, то в таком случае говорят, что к этим исходным данным построенный алгоритм неприменим или, иначе говоря, исходные данные являются для него недопустимыми. При необходимости решения задач для любого набора данных в алгоритм должны включаться дополнительные условия, обеспечивающие возможность его применения.

Рассмотрим в качестве примера алгоритм деления двух чисел. Пусть цель решения состоит в том, чтобы при делении двух чисел получить абсолютно точный результат, представленный или целым числом, или в виде десятичной дроби.

Если взять в качестве делимого и делителя числа 6,3 и 3, то очевидно, что алгоритм деления, выбранный для решения задачи, позволяет получить точный результат: 6,3:3=2,1.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Если же в качестве делимого и делителя соответственно взять числа 2,2 и 3, то, очевидно, будет получен следующий процесс деления:

2,2:3=0,7333…

Очевидно, что во втором примере процесс никогда не закончится, следовательно, воспользоваться для данного набора исходных данных приведенным выше алгоритмом нельзя, т.е., иначе говоря, указанные для второго случая исходные данные являются недопустимыми для сформулированного нами выше алгоритма деления.

Попытка оборвать процесс деления на некотором шаге дает возможность получить только приближенный результат. Соответственно, в алгоритме должно быть предусмотрено условие, определяющее необходимую степень точности получаемого результата. Таким образом, к исходным данным и результатам вычислений добавляются дополнительные условия проверки.

При построении алгоритма решения какой-либо конкретной задачи должно быть определено, что будет представлять результат решения (число, точку на плоскости и т.д.). Соответственно, выполнение этой операции должно обеспечиваться возможностями алгоритмической системы.

Заметим, что множество входных объектов (исходных данных) и множество результатов часто совпадают как в математических, так и в различных инженерных задачах. Например, в алгоритмах вычисления арифметических или алгебраических выражений и исходные данные и результаты — числа. Однако во многих задачах исходные данные и результаты могут оказаться объектами различной природы. Так, например, во многих задачах управления производством в качестве исходных данных, обеспечивающих решение задачи на этапе планирования, выступают числа, выражающие, например, производительность оборудования, его надежностные характеристики, эксплуатационные расходы и т.п. Результатом работы алгоритма может оказаться, например, информация, приводящая к решению о выборе нового оборудования.

Необходимым условием, учитываемым при создании и выборе алгоритмических систем, должно быть обеспечение возможности описания средствами одной системы некоторого числа различных алгоритмов для решения задач одного типа. Во многих случаях для получения решения одной задачи может быть использовано несколько алгоритмов. Поэтому на этапе алгоритмизации целесообразно не ограничиваться выбором одного алгоритма, обеспечивающего получение решения анализируемой задачи, а определить возможность использования здесь и других алгоритмов, выбрав в итоге из них лучший. При этом требуется обоснование, какой алгоритм считать лучшим. Это понятие не вполне определённое. Интуитивно ясно, что желательно сократить количество действий для достижения результата в процессе решения задачи и, по возможности, упростить сами эти действия.

Однако при этом любой вновь выбираемый или разрабатываемый алгоритм должен оставаться эквивалентным исходному. Сформулируем более чётко, что следует понимать под эквивалентностью алгоритмов. Будем называть два алгоритма эквивалентными, если:

-множество допустимых исходных данных одного из них является множеством допустимых исходных данных и для другого; из применимости одного алгоритма к каким-либо исходным, данным следует применимость и другого алгоритма к исходным данным;

-применение этих алгоритмов к одним и тем же исходным данным даёт одинаковые результаты.

Эквивалентными являются, например, алгоритмы для вычисления значения многочлена в некоторой точке x, заданные следующими формулами:

$$P_n(x) = a_n x^n + a_{n+1} x^{n+1} + a_{n+2} x^{n+2} + \dots + a_1 x^1 + a_0,$$

$$P_n(x) = \left(\left((a_n x + a_{n-1}) x + a_{n-2}\right) x + \dots + a_1\right) x + a_0.$$

Каждой из приведенных формул соответствует определенный алгоритм вычисления значений многочлена $P_n(x)$.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Для одного и того же набора значений коэффициентов обе формулы дают относительно одного значения xодинаковые результаты. Чтобы убедиться в этом, достаточно раскрыть скобки в формуле (2). Рассмотрим различия между приведёнными алгоритмами. По формуле (1) сначала вычисляется $x^2=x*x$, затем $x^3=x^2*x$, $x^4=x^3*x$, ... и т.д. Полученные результаты домножаются на соответствующие коэффициентыa_i: a_1*x, a_2*x^2, a_3*x^3… . Затем всё это складывается. Например, чтобы вычислить с помощью формулы (1) значение многочлена четвёртой степени $P_4(x)$ в точке x, необходимо проделать следующие вычисления: вычислить x^2, x^3, x^4 (на это уйдёт три операции умножения), затем получить значения a_1x, a_2x^2, a_3x^3, a_4x^4(это ещё четыре операции умножения) и, наконец, посчитать сумму

$$a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$$

(ещё четыре арифметические операции). На всё это потребовалось 11 арифметических операций.

Если же вычислять значение многочлена $P_4(x)$ в точке x по формуле (2), т.е.

$$P_4(x) = \left(\left(\left(a_4x + a_3\right)x + a_2\right)x + a_1\right)x + a_0.$$

то нетрудно подсчитать, что для получения результата потребуется выполнить 8 арифметических операций.

Итак, алгоритмы вычисления значения многочлена, определяемые формулами (1) и (2), эквивалентны. Но алгоритм, задаваемый формулой (2), требует меньше арифметических операций. Кроме того, он значительно удобнее при реализации на ЭВМ.

На выбор алгоритма может влиять также вид математической модели задачи, исходя из её конкретной постановки, в частности, исходя из задаваемых значений исходных данных, а также исходя из особенностей взаимосвязей между исходными и искомыми параметрами.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Проиллюстрируем влияние значений исходных данных на выбор алгоритма решения задачи, обратившись с этой целью к рассмотренному в конце п. 1.1 примеру решения квадратного уравнения. Там для нахождения значений корней был использован универсальный алгоритм решения полного неприведенного квадратного уравнения. Однако его применение при задании значения коэффициентаa равного 0 становится невозможным. В этом случае, при описании алгоритма для решения задачи на ЭВМ, должно быть определено направление его дальнейшего выполнения. Им может стать или завершение выполнения алгоритма или повторный ввод значенияa отличного от нуля и повторение решения. В обоих случаях в алгоритме должна быть предусмотрена выдача соответствующего сообщения пользователю.

Однако и при aравном нулю задача имеет решение. В этом случае математическая модель принимает вид $bx+c=0,$ т.е. становится линейной. Решение в этом случае может быть получено по алгоритму:

$$x = -\frac{c}{b}$$

Очевидно, что при описании алгоритма решения квадратного уравнения для ЭВМ более рациональным будет использование именно такого подхода для получения решения задачи приa =0.

Формально анализ оценки возможностиполучения решения квадратного уравнения может быть продолжен в описании алгоритма и для случаев равенства нулю значений переменных b и c. Не вдаваясь в детали можно отметить, что решение будет невозможным только приa иb одновременно получившим нулевые значения.

Влияние учета особенностей взаимосвязей между исходными и искомыми параметрами на построение модели и выбор алгоритма будет рассматриваться при изучении дисциплины “Основы теории принятия решений”.

Глава 2. Описание алгоритмов

2.1. Способы записи алгоритмов

Алгоритм становится алгоритмом тогда, когда он приобретает какую-либо форму представления, в соответствии с которой может быть выполнен. Выбор той или иной формы существенным образом зависит от того, на какого исполнителя алгоритм ориентирован (человек или автоматическое устройство). В настоящее время на практике применительно к различным ситуациям в основном используются следующие формы записи алгоритмов: *словесная, в псевдокоде, графическая, программная.*

Первая из них - словесная, является основной формой описания алгоритмов, традиционно принятой в математике. Она ориентирована на человека, изучающего правила решения той или иной задачи. Следует заметить, что термин “алгоритм” достаточно ограниченно применяется в классической математике. Для того чтобы убедиться в этом, достаточно обратиться к любому математическому справочнику, где обычно используются такие понятия как метод, способ, правило решения.

Остальные формы записи алгоритмовсвязаны в основном с обеспечением работы компьютеров и автоматических систем управления.

2.1.1. Характеристика словесного способа записи алгоритмов

Словесный способ записи алгоритмов представляет собой описание алгоритма в произвольной форме на естественном языке общения людей. При необходимости это описание может быть дополнено формулами.

В качестве примера рассматриваемого способа записи алгоритмов воспользуемся приводимым в справочниках по элементарной математике описанием правила деления простых дробей.

Чтобы разделить какое-либо число на дробь, нужно умножить это число на дробь, обратную делителю. Обратная дробь получается из исходной, если

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

у последней поменять местами числитель и знаменатель.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Описание дополняется следующей формулой:

$$\frac{a}{b}:\frac{c}{d}=\frac{a\cdot d}{c\cdot b}.$$

Очевидно, что данное описание достаточно для восприятия человеком. При этом предполагается, что для исполнителя является известным условие, в соответствии с которым ни одной из переменных в знаменателе дроби (для условий нашего примера переменным *b* и с) не может быть присвоено нулевое значение.

Возможный вариант словесного описания рассматриваемого алгоритма для ЭВМ может быть представлен в виде приведённой ниже последовательности записей.

1. Задать значения *a, b,*с, *d.*
2. Проверить значения *b* и *c* на неравенство нулю. Если $b=0$и(или) $c=0$, то вернуться к пункту 1.
3. Вычислить $m=a\cdot d$.
4. Вычислить $n=b\cdot c$.
5. Вычислить $z=m : n$.
6. Вывести значение *z.*

Словесный способ записи алгоритмов для автоматических устройств обработки информации и управления не нашел широкого распространения в силу того, что такие описания строго не формализуемы: допускают неоднозначность толкования отдельных предписаний, страдают многословностью записей.

2.1.2. Запись алгоритмов в псевдокоде

Понятие псевдокода связано с тем, что достаточно часто процесс программирования определяется как процесс кодирования. В этом случае принятая в некотором языке программирования система обозначений и правил определяется как код, обеспечивающий однозначное описание алгоритма.

Псевдокод также представляет определенную систему обозначений и

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

правил, предназначенную для единообразной записи алгоритмов.

Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, которым пользуется разработчик в своей повседневной жизни (в частности, русскому), поэтому 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 записи на языках программирования высокого уровня.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегает 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.

Единого формального описания псевдокода не существует, поэтому для описания алгоритмов используются различные псевдокоды, отличающиеся наборами служебных слов и основных (базовых) конструкций. В частности, примером псевдокода является школьный алгоритмический язык, изучаемый в курсе информатики в средней школе.

В предлагаемом пособии при рассмотрении псевдокода, как средства записи алгоритмов, ограничимся набором простейших разговорных конструкций, введя следующие обозначения типичных конструкций вычислительного процесса:

1. Ввод данных - ЧИТАТЬ.

2. Обработка данных:

V=выражение, где V - переменная; выражением задается правило

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

вычисления значения, которое далее будет присвоено переменной V. Чаще всего здесь записываются алгебраические или арифметические выражения.

3. Проверка условия:

ЕСЛИ *условие* ИДТИ К *N*

Если условие удовлетворяется, то выполняется переход к этапу с номером *N*. Если условие не удовлетворяется, то условимся, что осуществляется переход к следующему по порядку этапу.

4. Переход к этапу с номером *N:*

ИДТИ К *N*.

5. Вывод данных - ПИСАТЬ.

6. Конец вычислений

КОНЕЦ

При записи выражений (этап 2), которые задают правило вычислений, будет использоваться специальный знак присваивания =. Этот знак используется для изображения *операции присваивания,* смысл которой состоит в следующем. Пусть имеется предписание вида

$$y=A$$

(читается:*“у присвоитьА”),* где *у* - переменная, *А* - некоторое выражение.

Предписание означает следующее: выполнить все действия, предусмотренные выражением*А*, и полученный результат (число) считать значением (т.е. присвоить) переменнойу.Примеры записи операции присваивания:

$$x=12$$

$$\mathbf{y} = \ln\left(\sqrt{a^2 + 1}\right)$$

В левой части операции присваивания всегда должна стоять переменная.

Отметим важное свойство операции присваивания. Эта операция допускает случай, когда одна и та же переменная находится и слева и справа от знака присваивания. Так, например, можно написать

$$n=n+1$$

Такая запись означает, что сначала должна быть выполнена операция

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

сложения n+1, а затем полученная сумма присвоена переменной n в качестве ее нового значения. При этом старое значение nпропадает - “стирается”. После этой операции n будет иметь значение на 1 больше, чем перед ее выполнением.

Используя указанное свойство операции присваивания, можно избежать в записи алгоритмов введения избыточных вспомогательных переменных.

Из предыдущего изложения видно, что запись алгоритма в псевдокоде является промежуточным этапом при переходе от неформализованного описания алгоритма к программе для компьютера. Однако, следует заметить, что опытные программисты опускают этот промежуточный этап и сразу приступают к написанию программы.

Овладение описанием алгоритмов в псевдокоде может быть использовано в качестве начального этапа при изучении программирования. Поэтому в литературе рассматриваемую форму записи часто определяют как учебную программно-алгоритмическую.

Примеры записи алгоритмов в описанном псевдокоде будут рассматриваться в следующем параграфе параллельно с их графическим представлением.

Характеристика языков программирования, как средства описания алгоритмов, будет дана в пятой главе настоящего пособия.

2.1.3. Графический способ записи алгоритмов

При графическом представлении алгоритм изображается в виде последовательности связанных между собой линиями функциональных блоков, каждым из которых определяется выполнение соответствующего типа действий. Такое графическое представление алгоритма называется его *блок-схемой.* Блоки изображаются геометрическими фигурами. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

и т.п.) соответствует блок, представленный в виде определённой геометрической фигуры с записью внутри её соответствующего действия применительно к описываемому алгоритму. Блоки соединяются отрезками прямых (линиями переходов), которые определяют последовательность выполнения действий. При построении блок-схем направление сверху вниз принимается за основное. В сложных блок-схемах линии переходов для большей наглядности направления процесса обработки информации могут дополняться стрелками, а блоки нумероваться.

Правила построения блок-схем, виды и назначение используемых блоков определяется соответствующим государственным стандартом. Наиболее часто используемые блоки приведены в таблице 2.1.

Блок *«Процесс»* используется для описания операций обработки информации, в частности, описания подлежащей выполнению вычислительной операции, или последовательности операций. Содержание операций записывается внутри блока.

Блок *«Решение»* изображается ромбом с одним входом и двумя выходами. Функция рассматриваемого блока - выбор направления действий в зависимости от выполнения или невыполнения некоторых задаваемых условий. Условие записывается внутри ромба, например, в виде отношения $i<=m$. Выполнение этого условия (Да) соответствует одному из выходов ромба, невыполнение (Нет) - другому выходу.

При необходимости ГОСТом разрешается использование трех выходов. Однако без крайней необходимости использование такой конструкции нецелесообразно, так как при этом увеличивается возможность появления ошибок при описании алгоритмов.

Блок *«Модификация»* описывает управление циклом, т.е. многоразовым повторением одних и тех же операций. Внутри блока описывается закон изменения управляющего параметра цикла, например, в виде $i=i_n i_k \Delta i$. Здесь i_n- начальное значение параметра, i_k - его конечное значение, Δi - шаг изменения

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

параметра. Если шаг изменения параметра равен 1, то он может не указываться.

Блок *«Пуск, останов»* определяет начало и конец процесса обработки информации. Он используется так же при указании необходимости прерывания обработки данных.

Таблица 2.1

Название блока	Обозначение	Пояснение
Процесс		Вычислительное действие или последовательность вычислительных действий
Решение	Да Нет	Проверка условия
Модификация		Начало цикла
Пуск, останов		Начало, конец, останов, вход и выход
Предопределенный процесс		Вычисление по заранее созданной программе
Клавиатура дисплея		Ввод данных с клавиатуры дисплея
Ввод, вывод		Ввод или вывод данных
Вывод		Вывод результатов обработки данных: на экран, принтер

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Окончание таблицы 2.1

Соединитель		Указание на номер следующего блока либо от номера предыдущего
Переход		Переход к указанному блоку
Комментарий	Текст комментария	Пояснения действий, исходных данных, результатов, подпрограмм

Блоком *«Предопределенный процесс»* указывается использование в описанном алгоритме ранее созданных и отдельно описанных алгоритмов и подпрограмм. Внутри блока указывается имя алгоритма (подпрограммы), к которым надлежит обратиться и параметры, относительно которых они должны быть выполнены.

Для описания процессов ввода и вывода информации возможно использование нескольких блоков. Если устройство ввода или вывода в описании алгоритма не детализируется, то указанные операции на блок-схеме изображаются в виде параллелепипеда. При большой степени детализации используются блоки, привязанные к конкретным устройствам ввода-вывода (клавиатура, экран монитора, принтер). Внутри выбранного блока записываются имена входных или выходных переменных.

Блок *«Соединитель»* обеспечивает указания соединения между прерванными линиями связи блоков. Внутри блока указывается или номер блока, к которому должен быть осуществлен переход, или номер блока, от которого переход осуществляется.

Блок *«Переход»* указывает связи между разъединёнными частями алгоритма, представленными, например, на различных листах.

«Комментарий» используется в тех случаях, когда возникает необходимость в дополнительных пояснениях. Текст комментария помещают внутри

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

квадратной скобки, которая соединяется штриховой линией с блоком или отрезком между блоками, функции которых нужно прокомментировать.

Достоинством графического способа записи алгоритмов является наглядность изображения алгоритмов, что важно для представления алгоритмов сложной структуры. Такое представление алгоритмов существенно упрощает их восприятие программистами, за счёт чего в значительной степени сокращаются затраты труда и времени на написание программ. Естественно, что представление несложных для восприятия алгоритмов в виде блок-схем преследует в основном учебные цели.

2.2. Структура алгоритма. Понятие базовых алгоритмических структур

Любой алгоритм может рассматриваться как некоторая структура, состоящая из комбинации отдельных типовых структур. В теории алгоритмов доказана теорема, суть которой состоит в том, что *алгоритм для решения любой задачи можно составить только из структур: следование, ветвление, цикл.* Их называют базовыми алгоритмическими структурами.

Базовая структура *следование* образуется последовательностью действий, следующих одно за другим:

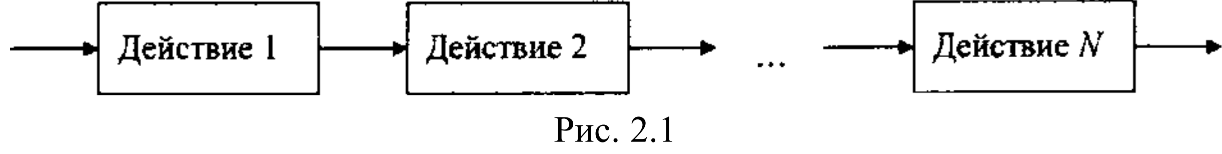


Рис. 2.1

Каждый блок может содержать в себе как простую команду, так и сложную структуру, но обязательно должен иметь один вход и один выход.

Ветвление - алгоритмическая альтернатива. Управление передается одному из двух блоков с зависимости от истинности или ложности заданного условия. Каждый из блоков (Серия 1, Серия 2), которому возможна передача управления, может содержать одно или некоторую последовательность (серию) действий. Последние могут быть линейными или, в свою очередь, раз-

ветвляющимися относительно некоторого, дополнительно заданного условия. После завершения серии происходит выход на общее продолжение алгоритма (рис. 2.2).

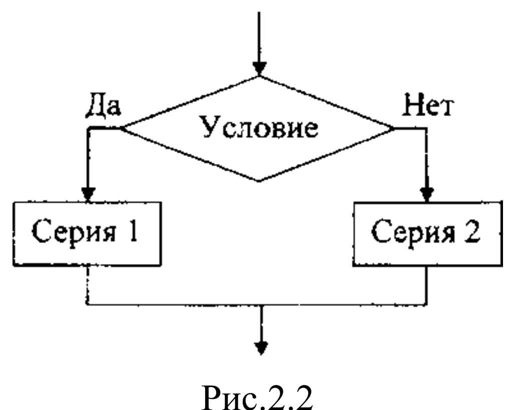


Рис.2.2

Возможно представление неполной формы ветвления, когда по ветви «Нет» осуществляется выход на общее продолжение алгоритма без выполнения каких-либо промежуточных операций:

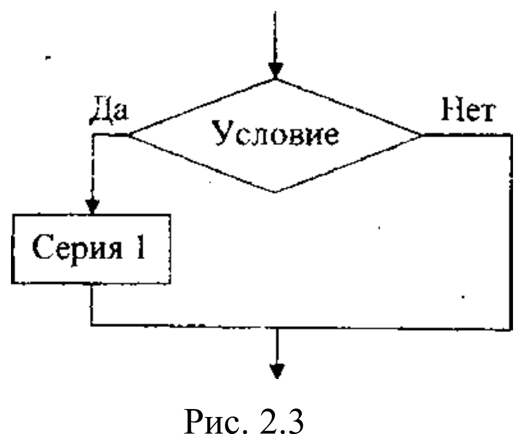


Рис. 2.3

Цикл - повторение некоторой группы действий по задаваемому условию. Циклический процесс может быть описан двояко либо посредством использования блока "Решение", либо обращением к блоку "Модификация". В первом случае описание более наглядно, что в частности, используется в учебных целях. Во втором - обеспечивается большая компактность записи алгоритма.

Различаются два типа циклов. Первый - *цикл с предусловием (цикл - пока):*

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

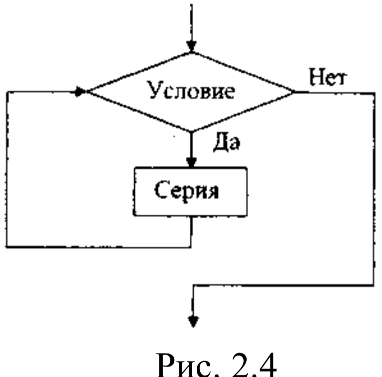


Рис. 2.4

Пока условие истинно, выполняется серия, образующая тело цикла.

Второй тип циклической структуры - *цикл с постусловием (цикл-до):*

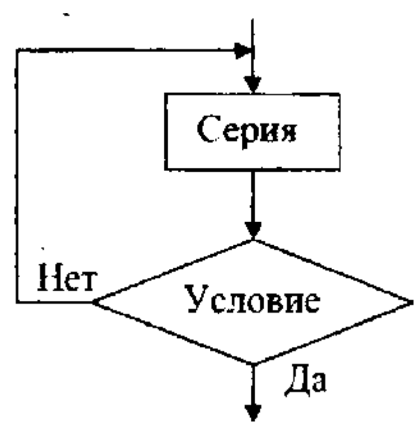


Рис. 2.5

Здесь тело цикла предшествует условию цикла. Тело цикла повторяет свое выполнение, если условие ложно. Повторение кончается, когда условие станет истинным.

Теоретически необходимым и достаточным является лишь первый тип цикла - цикл с предусловием. Любой циклический алгоритм можно построить с его помощью. Это более общий вариант цикла, чем цикл-до. В самом деле, тело цикла-до хотя бы один раз обязательно выполнится, так как проверка условия происходит после завершения его выполнения. А для цикла-пока возможен такой вариант, когда тело цикла не выполнится ни разу. Поэтому в любом языке программирования можно было бы ограничиться толь-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ко циклом-пока. Однако, в ряде случаев, применение цикла-до оказывается более удобным, и поэтому он находит практическое применение.

Начальное ознакомление с графическим представлением циклов обычно производится на основе блока «Решение». Освоение этого подхода существенно упрощает понимание описания циклов с помощью блока «Модификация».

Возможные базовые алгоритмические структуры приведены в таблице 2.2. Примеры описания алгоритмов применительно к ним будут рассмотрены в следующем разделе.

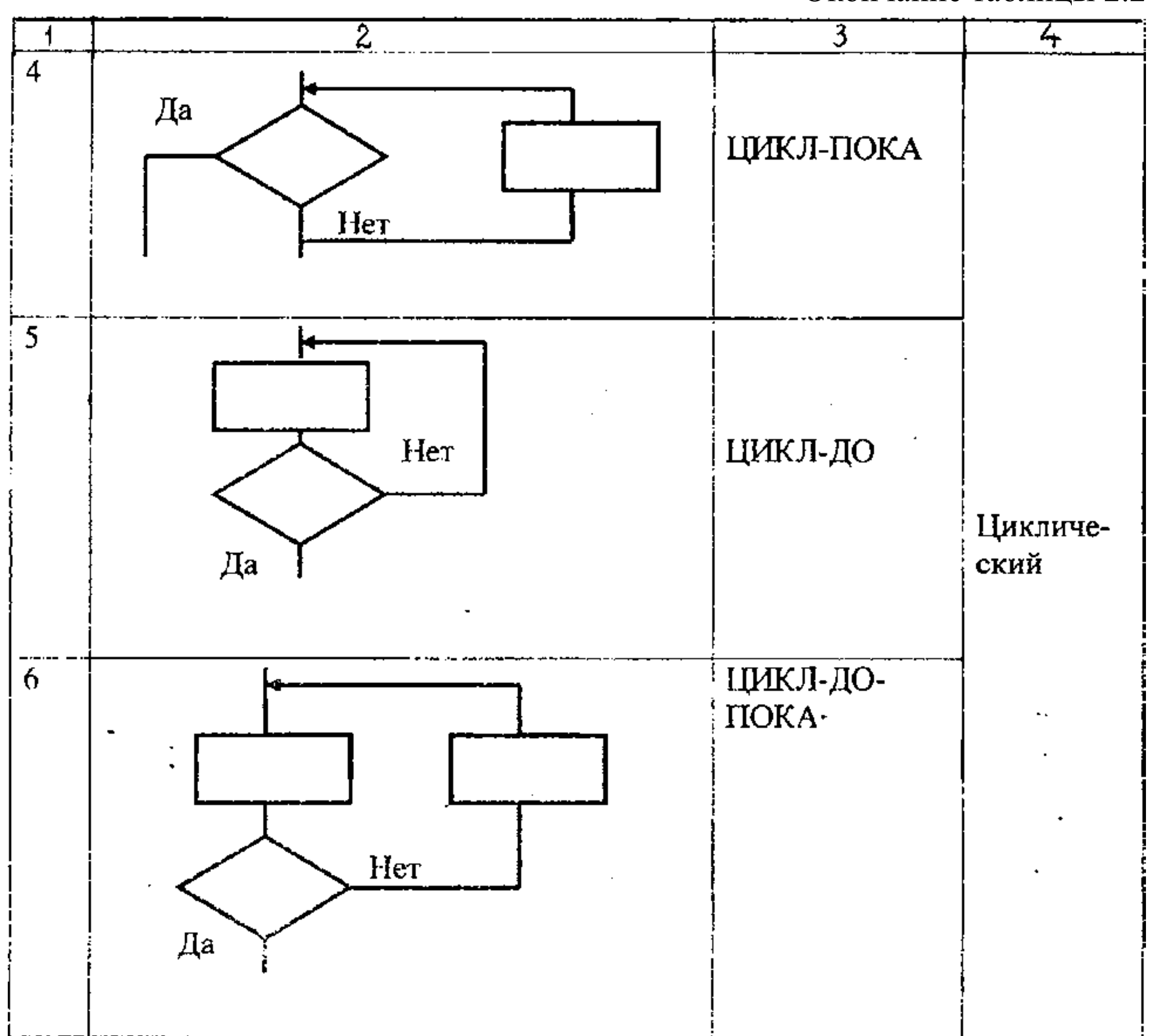
Таблица 2.2

№ п.п.	Обозначение на схеме	Наименование	Тип алгоритма
1	2	3	4
1		СЛЕДОВА-НИЕ	Линейный
2	Да Нет	ЕСЛИ-ТО	Разветв-ляющийся
3	Да Нет	ЕСЛИ-ТО-ИНАЧЕ	

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Окончание таблицы 2.2



1	2	3	4
4	Да Нет	ЦИКЛ-ПОКА	Циклический
5	Нет Да	ЦИКЛ-ДО	
6	Нет Да	ЦИКЛ-ДО-ПОКА	

2.3. Описание линейных алгоритмов

Суть практически любого линейного алгоритма очевидна из его словесного или словесно-формульного описания. Целесообразность изображения алгоритма такого типа в виде блок-схемы может обеспечить его большую наглядность, например, для программиста. Чаще линейные схемы являются составными частями блок-схем разветвляющихся и циклических алгоритмов.

Пример 2.1. Требуется записать в графической и псевдокодовой формах алгоритм вычислений применительно к заданной ниже формуле:

$$y=(ax^2+b)*(ax+1)$$

При составлении требуемых алгоритмических записей примем, что правила выполнения арифметических операций, в частности, сложения, умножения и деления понятны вычислителю. С учётом этого графическая и псевдокодовая формы записи могут иметь вид (рис. 2.6).

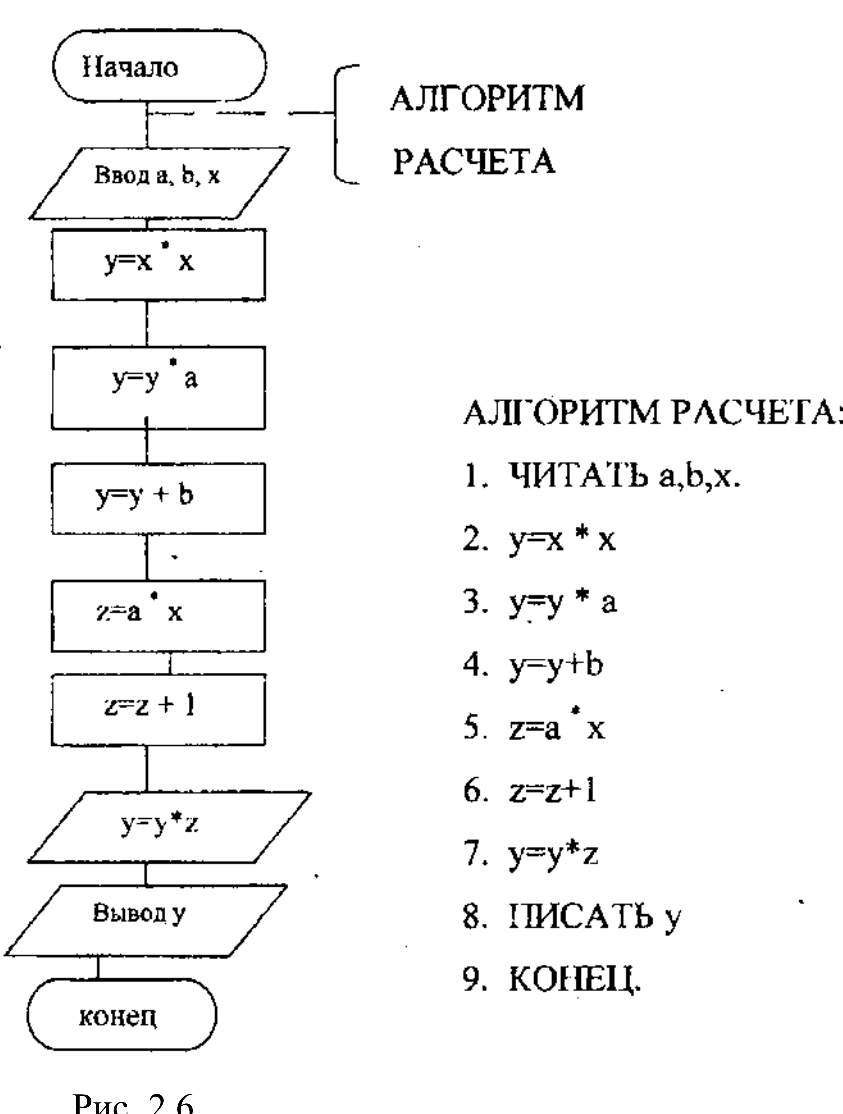


Рис. 2.6

Из сравнения представленных на указанном рисунке изображений можно сделать вывод о том, что графическая и псевдокодовая формы записи линейных алгоритмов сходны. Следует особое внимание обратить на первое и последнее предписания в этих записях, обеспечивающие задания значений исходных данных и выдачу полученного результата. При всей кажущейся излишней строгости этих описаний они имеют первостепенное значение в

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

алгоритмах, адресуемых исполнителям - компьютерам. Формулируя эти предписания, автор алгоритма должен четко определить для себя, что в этой задаче «дано», а что требуется «получить». Или, говоря иными словами, какие величины являются носителями исходных значений (входные величины или аргументы), а какие - носителями значений результатов (выходные величины или результаты). В приведенной выше записи вычислительного алгоритма входными величинами являются a, b, x, а выходной - y.

Отметим другое важное обстоятельство. Несмотря на то, что в приведенной записи алгоритма ни в одном из предписаний ничего не сообщается о том, к какому предписанию следует переходить в следующий момент, представляется вполне естественным, что всюду предполагается переход к предписанию, непосредственно идущему за выполняемым. Именно этого правила и следует придерживаться, т.е. если при исполнении текущего предписания алгоритма никак не определяется номер очередного предписания, то это будет означать автоматический переход к предписанию, следующему заданным в порядке возрастания номеров.

В данном примере за счет применения операции присваивания удалось обойтись лишь двумя переменными y и z, которые используются для хранения всех промежуточных результатов, последовательно получаемых в процессе вычислений. Этим обеспечивается меньшая загрузка памяти ЭВМ и сокращается объём работы программиста при написании программы.

Следует обратить внимание на то, что и для описания ввода исходных данных и для описания вывода информации использован блок в виде параллелограмма. Напомним, что такое изображение указанных процессов универсально относительно всевозможных используемых устройств ввода-вывода. В последующих примерах ограничимся использованием этого блока, не забывая при этом, что в практике описания алгоритмов чаще указываются конкретные устройства.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Составные линейные алгоритмы предусматривают обращения к модулям. Работа с модульными структурами будет рассмотрена ниже.

ЗАДАНИЕ. Выбрать из возможных и описать наиболее приемлемый для реализации на компьютере алгоритм решения системы двух линейных уравнений с двумя переменными.

2.4. Описание разветвляющихся алгоритмов

При решении инженерных задач широко применяются разветвляющиеся алгоритмы. При этом вызывает особую сложность организация переходов от предписания к предписанию. Дело в том, что в разветвляющихся алгоритмах принцип линейного автоматического перехода от предписания к предписанию в порядке естественного возрастания их номеров уже не является всеобщим, так как время от времени возникает потребность перехода к предписанию с произвольными, а необязательно следующими по порядку номерами. В блок-схемах алгоритмов все переходы организуются одним способом - с помощью стрелок. В псевдокодовой записи алгоритмов наряду с механизмом линейной передачи управления используются также и предписания специального вида для организации так называемых безусловных и условных переходов.

Предписание безусловного перехода имеет вид:

ИДТИ К N,

где N - номер одного из предписаний в записи алгоритма. Само по себе предписание безусловного перехода никаких действий не вызывает, кроме передачи управления предписанию, номер которого явно указан в предписании безусловного перехода.

Предписание условного перехода соответствует блоку «решение» в схемах алгоритмов и имеет вид:

ЕСЛИ *P* ИДТИ К *N*.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Здесь *P* - проверяемое условие, N - номер предписания, к которому происходит переход, когда условие *P* соблюдается (истинно). Если условие *P* ложно, происходит переход к предписанию, расположенному вслед за предписанием условного перехода в порядке возрастания номера. Рассмотрим пример операции условного перехода:

3.ЕСЛИ x<0 ИДТИ К 6.

Условием *P* в данном случае является неравенство x<0, которое, например, при x=1 будет ложным. Следовательно, данное предписание осуществит переход (или, как говорят еще, передаст управление) не на предписание N=6, а на предписание, следующее за предписанием с номером 3, т.е. на предписание 4. Таким образом, срабатывает следующий принцип: если при исполнении текущего предписания алгоритма номер очередного предписания не определяется по условию, то подобно тому, как это принято в линейных алгоритмах, происходит автоматический переход к предписанию, следующему заданным в порядке возрастания номеров.

Пример 2.2. Требуется составить запись алгоритма решения уравнения $ax=2$, где a - произвольное действительное число.

В данном случае суть алгоритма сводится к проверке условия $a \neq 0$. Если оно соблюдается, то решение находится тривиально: $x=2/a$\ если же $a=0$, ни одно текущее значение x не удовлетворяет решению. Схема алгоритма изображена на рис. 2.7.

На предложенном примере рассмотрим детально процесс псевдокодовой формы записи алгоритма. Вслед за чтением значения исходного данного a нужно сразу же приступить к проверке условия $a \neq 0$. Появляется следующее начало будущего алгоритма:

АЛГОРИТМ решения уравнения

1. ЧИТАТЬ a

2. ЕСЛИ $a \neq 0$ ИДТИ К...

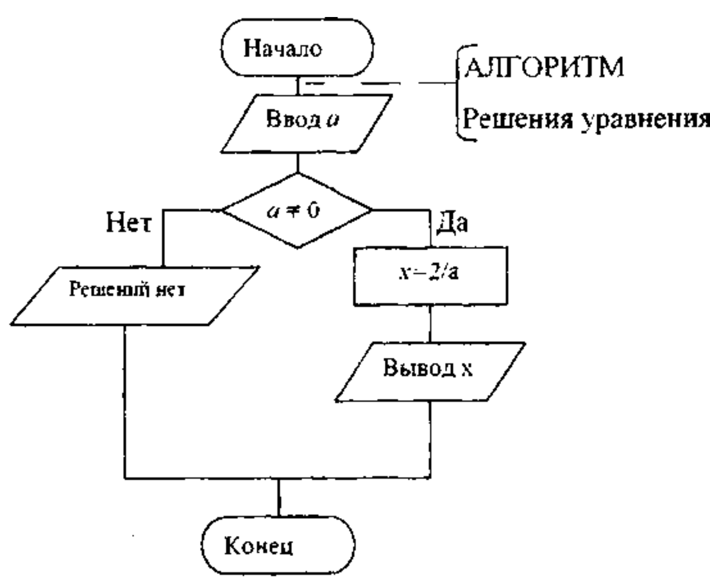


Рис. 2.7

Многоточие здесь означает, что нам пока не известен номер предписания, к которому нужно будет переходить при выполнении условия$a \neq 0$. Однако на основании правил выполнения предписания условного перехода нам известно, что в случае несоблюдения условия$a \neq 0$ (т.е. соблюдения условия $a=0$) исполнитель перейдет к предписанию с номером 3. Рассмотрим в первую очередь эту логическую ветвь. Учитывая то, что при$a=0$ решения нет, предписание 3 выводит это текстовое сообщение на экран дисплея, либо на печать. Во всех таких случаях выводимый текст берется в кавычки. После текстового сообщения будет произведен безусловный переход к окончанию алгоритма (номер его пока неизвестен), т.е.

3. ПИСАТЬ «*решений нет*», ИДТИ К...

В этой записи предписание безусловного перехода размещено в одной строке вслед за предписанием чтения с целью экономии строк, как это обычно и применяется в стандартных алгоритмических языках.

Вернемся к логическому условию с номером 2. Если оно выполняется (т.е.соблюдается условие $a\neq0$), то решением уравнения будет $x=2/a$.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Предписанию этого решения можно присвоить следующий номер 4 и тем самым исключить многоточие в предписании 2.

Окончательная запись алгоритма:

АЛГОРИТМ решения уравнения

1. ЧИТАТЬ a
2. ЕСЛИ$a \neq 0$ ИДТИ К 4
3. ПИСАТЬ *«решения нет»*; ИДТИ К 6
4. x=2/a
5. ПИСАТЬ x
6. КОНЕЦ.

Предписание «конец» получило номер 6, поэтому строка 3 заканчивается предписанием ИДТИ К 6. А строке 5 такого предписания не потребовалось, так как переход к окончанию алгоритма в данном случае произойдет автоматически.

Напервых порах для тренировки необходимого навыка понимания процесса исполнения алгоритма полезно обращаться к специальной процедуре проигрывания алгоритмической записи. Процесс проигрывания алгоритмической записи рассмотрим на примере только что описанного алгоритма вычисления x.

Суть этого процесса состоит в том, что выбираются конкретные исходные данные и описанный алгоритм педантично исполняется со строгим исполнением всех содержащихся в его записи предписаний. Каждое выполняемое предписание фиксируется в специальном бланке, форма которого для рассматриваемого примера показана в табл. 2.3. В таблице отведены столбцы для номеров предписаний (шаги алгоритма), для всех фигурирующих в описании величин (в данном случае это переменные a и x). Отдельная графа отведена для текущих значения переменной a: 5и 0. Из первого столбца таблицы хорошо видно, что получение результата решения задачи в каждом из

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

этих двух случаев алгоритм обеспечивает разными путями: в первом случае действуют предписания 1,2,4,5,6, во втором - 1,2,3,6.

Таблица 2.3

Шаги алгоритма	Аргумент	Результат	Пояснения
	a	x	
1 2 4 5 6	5	 0,4 0,4	ЧИТАТЬ *a* $5 \neq 0$ (истинно) x=2/a ПИСАТЬ x КОНЕЦ
1 2 3 6	0	 «Решения нет»	ЧИТАТЬ *a* $0 \neq 0$ (ложно) ПИСАТЬ «решения нет» КОНЕЦ

Решение многих практических задач требует применения в алгоритмах двух и более ветвлений. Проиллюстрируем это на следующем примере.

Пример 2.3. Требуется составить запись алгоритма нахождения наименьшего значения из трех переменных: *a,* b и c. Смысл задачи состоит в нахождении значения y=min*(a,*b, c).

Применительно к поставленной задаче наглядна зависимость выбора возможного алгоритма от исполнителя. Для человека решение данной задачи можно задать следующей последовательностью действий: расположить значение переменных в ряд, выбрать среди них меньшее. Для компьютера алгоритм получения решения данной задачи может быть следующим. Вначале в качестве наименьшего выбирается текущее значение любой из трех переменных, например, *a.* Тогда y=a. Естественно, что это пока предположение, которое следует проверить путем сравнения y со значением следующей пере-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Меной b. Если окажется, что условие $y \le b$ не выполняется, то предположение о том, что $a \le b$, неверно и необходимо присвоить промежуточному результату y значение b $(y = b)$. В противном случае $(b > a)$ сохраняется прежнее значение y. Теперь остается сравнить промежуточный результату со значением с и b зависимости от исхода сравнения либо оставить без изменения значение y, либо заменить его на значение c $(y = c)$. Это и будет выходной результат исполнения алгоритма. Псевдокодовая и графическая записи алгоритма показаны на рис. 2.8.

АЛГОРИТМ определения наименьшего значения

1. **ЧИТАТЬ** a, b, c
2. $y=a$
3. **ЕСЛИ** $y \le b$ **ИДТИ К 5**
4. $y=b$
5. **ЕСЛИ** $y \le c$ **ИДТИ К 7**
6. $y=c$.
7. **ПИСАТЬ** y
8. **КОНЕЦ.**

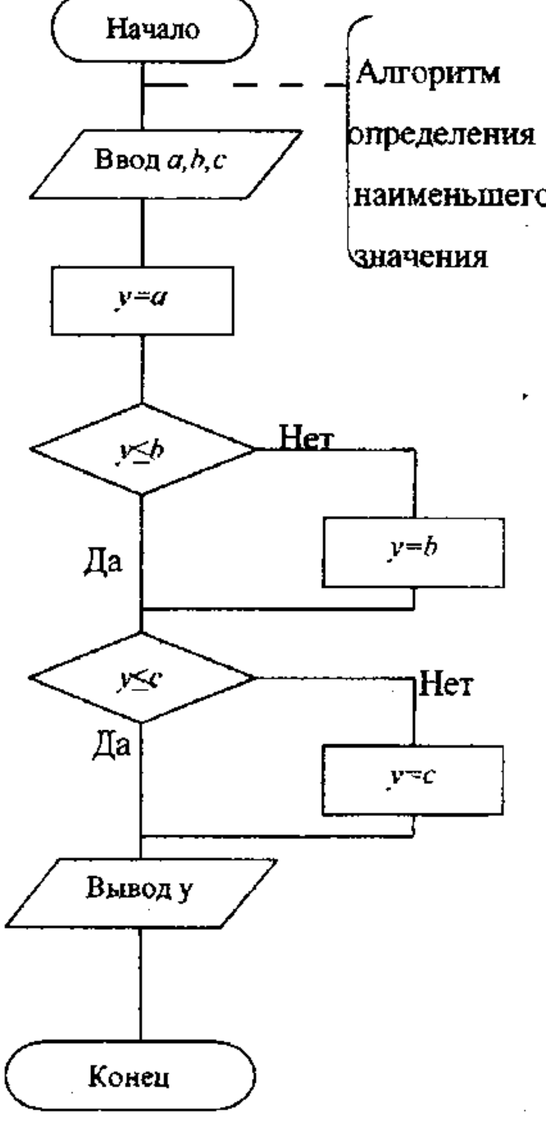


Рис. 2.8

В таблице 2.4 показан пример исполнения данного алгоритма при значенияхa=3, b = 5, c = 2.

При взгляде на эти числа для человека очевидно, что наименьшим из них будет *с-2.* Чтобы научить машину решению таких задач, нужно ввести а нее программу написанную по приведенному алгоритму. Данный пример поучителен при изучении программирования.

Таблица 2.4

Шаги алгоритма	Аргумент			Результат	Пояснения
	а	в	с		
1	3	5	2		ЧИТАТЬ а,в,с
2				3	
3					$3 \le 5$ (ИСТИННО)
5					$3 \le 2$ (ЛОЖНО)
6				2	
7					ПИСАТЬ 2
8					КОНЕЦ

Во-первых, он показывает, как машину можно научить решать логические задачи (т.е. научить мыслить). Во-вторых, он свидетельствует о том, что увеличение количества переменных, из которых нужно выбрать наименьшее (либо наибольшее) значение, обеспечивается выполнением одинаковых действий, поскольку все этапы анализа в алгоритме одинаковы. Действительно, предписания 3 и 4 идентичны предписаниям 5 и 6 - оба эти этапа алгоритма включают предписания условного перехода и присвоения. Как будет показано в следующем разделе (п.2.5), логический анализ множества переменных можно свести к одному этапу алгоритма, включающему всего два указанных предписания.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Задание. Описать алгоритм получения численного решения дифференциального уравнения y'=xy методом Эйлера в промежутке (0,1) при начальных значениях x_0=0, y_0=1.

2.5. Описание циклических алгоритмов

Циклические алгоритмы широко применяются для решения инженерных задач, в которых процесс обработки информации многократно повторяется по одним и тем же правилам. Они позволяют существенно сократить объем программ. Рассмотрим применение циклов в алгоритмах некоторых характерных задач.

Пример 2.4. Построить алгоритм вычисления значений функции $y=a^i$ при изменении натуральных чисел i от начального значения $i_н$=1 до конечного значения $i_к$=n с шагом, равным 1,

Для заданных условий задача может заключаться в заполнении значениями *y* следующей таблицы:

Таблица 2.5

i	1	2	3	...	n-1	n
y	y_1	y_2	y_3	...	y_{n-1}	y_n

Порядок решения этой задачи следующий. Вначале нужно присвоить переменной i значение i= 1, рассчитать по заданной формуле результату и выдать его на печать, либо экран терминала. Затем можно задать следующее значение переменной i= 2, рассчитать результат y_2, выдать его на печать и так далее, включая i=n и результат y_n. Однако такие предписания, допустимые для человека, не годятся для компьютера. Действительно, если, например, известно, что *n*=10, человек, рассчитывая очередные значения$y_2,\ldots,y_9$и y_{10}, автоматически выполняет в уме логические операции типа $2 \leq 10$, $3\leq 10$,..., $9 \leq 10$, $10 \leq 10$, не замечая этого. Компьютер же нужно научить этому

логическому действию, которое должно им повторно выполняться в каждом цикле заполнения очередной клетки таблицы результатов. Графическая запись алгоритма показана на рис. 2.9

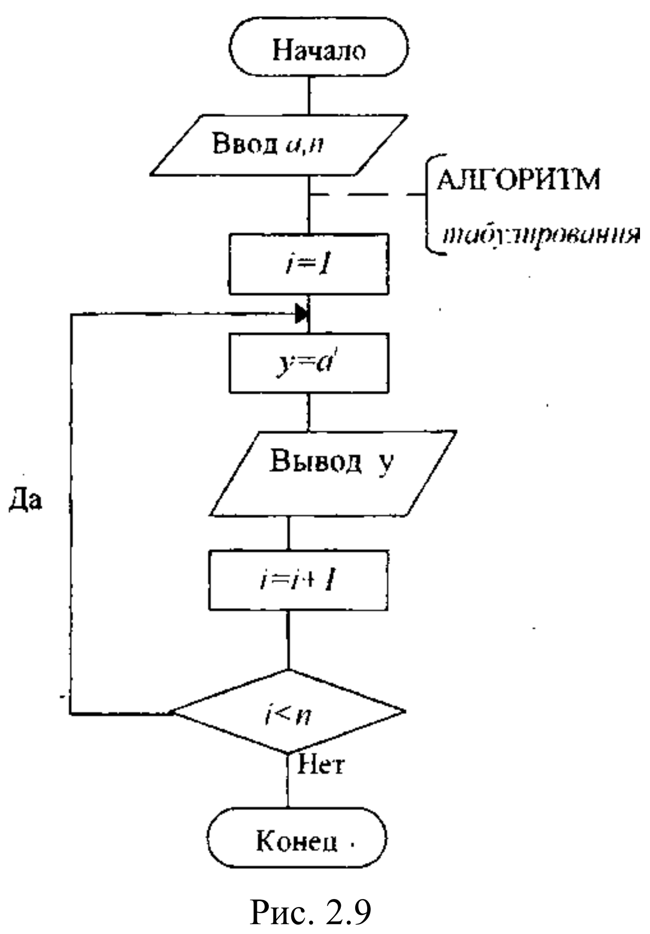


Рис. 2.9

Название алгоритма соответствует его назначению - табулировать функцию, т е. вычислять значение *y* при изменении аргумента*i* от $i_н=1$ до $i_к=n$ с шагом Δi=1. Предписание $i=i+1$ предназначено для определения следующего значения аргумента, а предписание условного перехода служит для принятия решения либо на выход из цикла, либо на повторение предписаний, которые определяют вычисление очередного значения результата и подготавливают следующее значение аргумента. Переход с конца циклического участка в его начало показывается на графической схеме стрелкой.

Таблица 2.6

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Шаги алгоритма	Переменные		Результат	Пояснения
	a	n	y	
1	2	5		ЧИТАТЬ a,n
2				i=1
3			2	
4	Цикл 1			ПИСАТЬ 1,2
5				i=2
6				2≤ 5 (ИСТИННО)
7			4	
8	Цикл 2			ПИСАТЬ 2,4
9				i=3
10				3≤5 (ИСТИННО)
11			8	
12	Цикл 3			ПИСАТЬ 3,8
13				i=4
14				4≤5 (ИСТИННО)
15			16	
16	Цикл 4			ПИСАТЬ 4,16
17				i=5
18				5≤5 (ИСТИННО)
19			32	
20	Цикл 5			ПИСАТЬ 5,32
21				i=6
22				6≤5 (ЛОЖНО)
23				КОНЕЦ

В табл. 2.6 приведен пример исполнения данного циклического

алгоритма при значениях *a=2* и *n=5.* Из таблицы видно, что количество шагов циклического алгоритма больше числа шагов при его линейной записи, в частности из-за применения операций сравнения. Отсюда и время выполнения циклического алгоритма прямо пропорционально количеству его циклов. Вместе с тем, если количество предписаний данного циклического алгоритма равно 7, то при линейной его записи для *n*=5 количество предписаний совпадает с числом шагов и равно 23. Можно подсчитать, что при n= 6 в линейной записи было бы 23+4=27 предписаний, а при *n*=20 было бы 23+60 =83 предписания, тогда как в циклической записи количество предписаний алгоритма не зависит от числа циклов и при *n*= 20 по-прежнему остаётся равным 7.

Циклический алгоритм в данном примере можно записать более компактно, если предписания присваивания и условного перехода заменить в псевдокодовой записи предписанием ДЛЯ-СЛЕДУЮЩИИ. Этому предписанию в графической записи соответствует блок «модификация».

Указанное предписание используется в алгоритмах многих задач. Поэтому следует подробно рассмотреть его содержание.

Общий вид предписания:

ДЛЯ $V = a_1$ *К* a_2 *ШАГ*a_3

СЛЕДУЮЩИЙ *v*,

где *v* - управляющая переменная цикла; a_1 a_2, a_3 - начальное, конечное значения и шаг изменения управляющей переменной цикла (в общем случае ими могут быть любые арифметический выражения). В рассматриваемом *примере* $a_1=i_H=1$, $a_2=i_K=n$, $a_3=\Delta i=1$. *Если* $a_3=1$, *то конструкцию ШАГ* a_3 *можно опустить. Предписания, расположенные между предписаниями ДЛЯ и СЛЕДУЮЩИЙ, образуют тело цикла и выполняются многократно.*

Выполнение цикла, образованного предписаниями ДЛЯ-СЛЕДУЮЩИЙ осуществляется в такой последовательности: переменной *v* присваивается начальное значение a_1 и затем оно сравнивается с конечным значением a_2.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Если при положительном приращении a_3 переменной v удовлетворяется условие $v \le a2$ или при отрицательном изменении a_3 удовлетворяется условие $v \ge a_2$, то выполняются предписания, образующие тело цикла, а по предписанию СЛЕДУЮЩИЙ осуществляется возврат к началу цикла. После этого значение v изменяется на шаг a_3, т.е. $v=v+a_3$, и снова проверяется заданное условие. Если оно удовлетворяется, то операторы тела цикла выполняются вновь. В противном случае происходит выход из цикла и переход к очередному предписанию.

Модифицированные варианты графической и псевдокодовой записи алгоритма для примера 2.4 показаны на рис. 2.10.

АЛГОРИТМ *табулирования*

1. **ЧИТАТЬ** a,n
2. **ДЛЯ** $i=1$ К n
3. $y=a^i$
4. **ПИСАТЬ** i,y
5. **СЛЕДУЮЩИЙ** i
6. **КОНЕЦ**

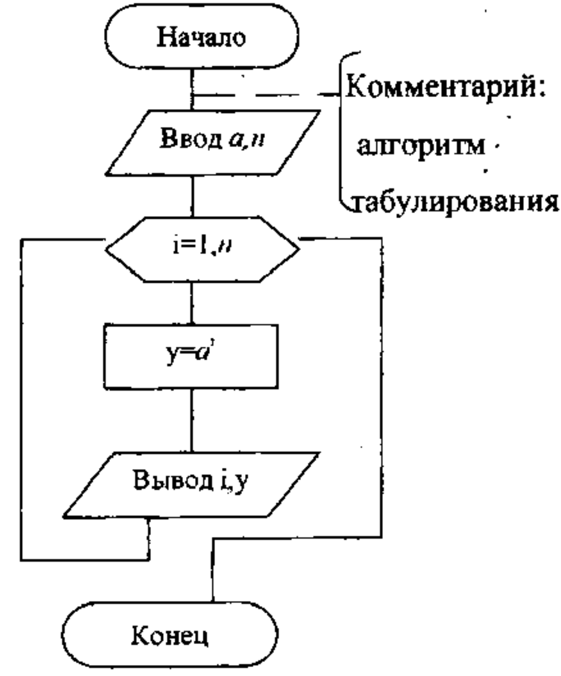


Рис. 2.10

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Задание. Составить алгоритм вычисления значений переменной y_i, относительно заданного значения x для I, изменяющегося с шагом, равным 1, от I, равного от 5 до 10:

$$y_i = \frac{x^i}{i}$$

В рассмотренном выше примере аргумент представлял собой целое число. Более общая задача построения циклического алгоритма табулирования функции имеет место при действительном аргументе, например, x, который изменяется от $x_н$ к $x_к$ с шагом Δx, не равным 1.

ПРИМЕР 2.5.Построить алгоритм табулирования функции $y=a^x$ при изменении действительного аргумента x от начального значения $x_н$ до конечного значения $x_к$ с шагом Δx. Отличие данного алгоритма от ранее рассмотренного состоит в замене целых чисел i на действительные x, а также в изменении содержания операций присваивания:

Было в примере 2.4:	Есть в примере 2.5:
2. $i=1$	2.$x=x_н$
…	…
5. $i=i+1$	5. $x=x+\Delta x$

В модифицированной графической записирассматриваемой задачи в бло-

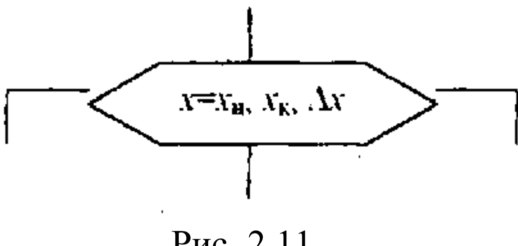


Рис. 2.11

ке модификации указывается (рис 2.11):

В свою очередь, в модифицированной псевдокодовой записи следует дополнить предписание 2 значением шага Δx:

2. ДЛЯ $x=x_H$, $К$ $x_к$ ШАГ Δx.

Все остальные предписания в алгоритме данного примера остались таки-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ми же, как и в предыдущем.

Во многих практических задачах, связанных с обработкой числовой информации, приходится иметь дело с конечными числовыми последовательностями - массивами. Элементы массива располагаются последовательно в памяти компьютера. Каждому элементу массива могут быть присвоены различные значения, в том числе и одинаковые для всех элементов. Все элементы одного массива обозначают одной буквой с индексом или индексами. Например, элементы одномерного массива:

$$a_1\ a_2, a_3,\ \dots, a_i \dots, a_m.$$

Число m в данном случае называют длиной массива.

Поскольку место каждого элемента массива определяется в последовательности или в таблице по индексам, то в описании алгоритмов, связанных с массивами, индексы играют существенную роль. При описании алгоритмов индексы могут оказаться в роли переменных. Например, предписание $C_j = a_i + 7$ $(j\text{-}1, \dots,\ m)$ указывает на выполнение m действий с различными значениями a_j.

Для обработки массивов, как правило, используются циклические алгоритмы.

ПРИМЕР 2.6. Построить алгоритм табулирования функции:

$$y_j = a^{x}{}_{j}\ , где\ j = 1, 2, \dots, 20.$$

В данном примере x - массив действительных чисел, каждый элемент которого никаким образом не связан с любым другим элементом.

Содержимое данного алгоритма во многом соответствует содержанию ранее рассмотренных циклических алгоритмов. Действительно, так же как и ранее следует задать определенное значение показателя степени и вычислить очередное значение функции y_j. Эти вычисления повторить в условиях данного примера 20 раз, убеждаясь при каждом повторении (цикле), что не

все элементы массива x_j ещё использованы. Однако, при обработке массивов

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

имеют место также и следующие специфические действия:

-необходимость описания размерности массива;

-организация цикла по изменению индекса j элемента массива x_j, но не по значению x_j.

Этоочень важные особенности циклических,алгоритмов обработки массивов. Они обусловлены следующим: элементы массива не упорядочены по своим значениям - они упорядочены только по индексам. Для сравнения вспомним, что в предыдущих циклических алгоритмах и переменная i, и переменная x были упорядочены по своим значениям.

Графическая и псевдокодовая записи данного алгоритма показаны на рис. 2.12.

АЛГОРИТМ *табулирования*

1. **РАЗМЕР** x (20)
2. **ЧИТАТЬ** a
3. **ДЛЯ** j= 1 **К** 20
4. **ЧИТАТЬ** x_j
5. $y_j = a^{x_j}$
6. **ПИСАТЬ** x_j y_j
7. **СЛЕДУЮЩИЙ** j
8. **КОНЕЦ**

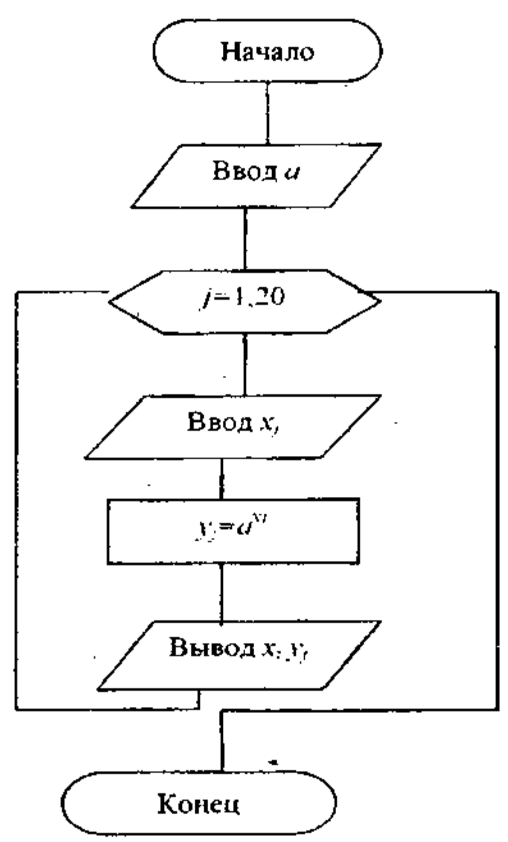


Рис. 2.12

В этом примере можно видеть, что графической форме записи не хватает

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

средств для описания всех необходимых предписаний алгоритма. Так, она не позволяет описать массив данных и не содержит однозначных указаний на то, каким образом выбираются значения элементов массивов в очередном цикле алгоритма. Поэтому графическую запись следует рассматривать только как концептуальную, наглядно отражающую принципы построения алгоритма. Псевдокодовая форма обеспечивает полную запись алгоритма.

Выбор значения очередного элемента x_j массива данных осуществляется с помощью предписания 4. Остальные предписания (2,3,5,6,7 и 8) точно такие же, как и у алгоритма, построенного в примере 2.5.

Важным свойством циклической организации алгоритмов является его общность для различных задач. Рассмотрим задачу, которая по содержанию ничего общего не имеет с табулированием функции, и убедимся, что алгоритм ее решения очень схож с рассмотренным выше алгоритмом.

ПРИМЕР 2.7. Требуется построить алгоритм нахождения наименьшего из значений элементов массива С.

Идея решения этой задачи описана в предыдущем разделе (п.2.4) при рассмотрении примера 2.3.

В записи алгоритма будет использована универсальная процедура ввода-вывода массивов. Она заключается в организации циклического чтения элементов массива, начиная с первого $C_1(j=1)$ и до конечного $C_m(j=m)$, где m - назначенная пользователем алгоритма длина массива.

Графическая и псевдокодовая записи алгоритма показаны на рис. 2.13.

*АЛГОРИТМ*поиска наименьшего значения

ЧИТАТЬ *m*

РАЗМЕР С *(m)*

ДЛЯ$j=1$ Кm

ЧИТАТЬ C_j

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

5. СЛЕДУЮЩИЙ j

6. $y=C_1$

7. ДЛЯ $j=2$ К m

8. ЕСЛИ $y \leq C_j$ ИДТИ К 10

9. $y=C_j$

10. СЛЕДУЮЩИЙ j

11. ПИСАТЬ y

12. КОНЕЦ

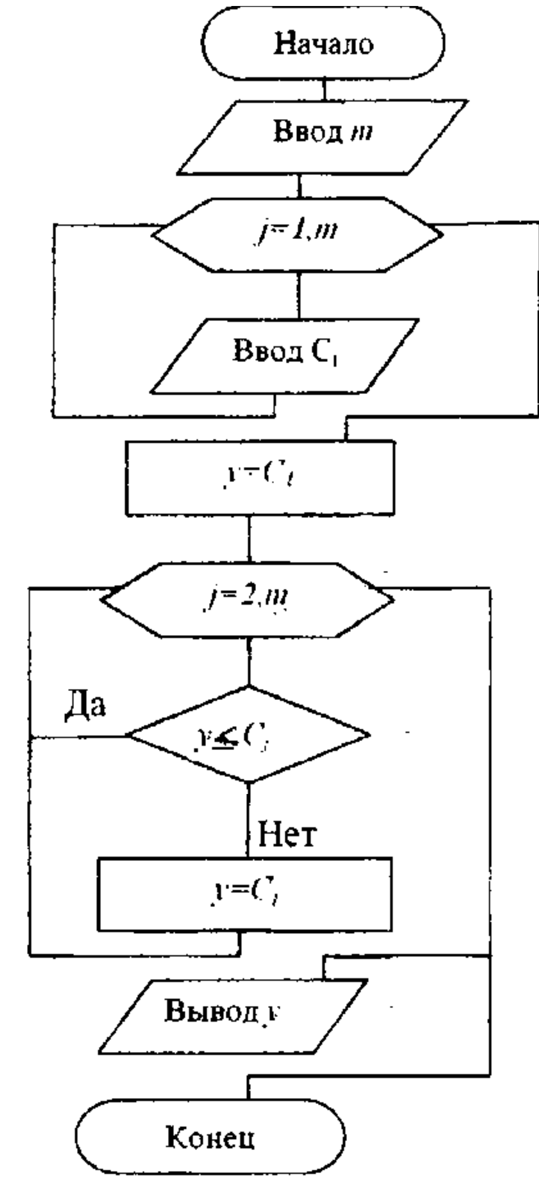


Рис. 2.13

В псевдокодовой форме записи алгоритма циклическое чтение элементов массива обеспечивается предписаниями 2,3,4 и 5.

После ввода данных переменной y присваивается значение первого элемента C_1 массива (предписание 6). Затем с помощью цикла, образованного предписаниями 7,8,9,10, последовательно сравнивается значение у со значениями элементов C_2, C_3 ...,C_m.Если выяснится, что какое-либо из значений этих элементов (напримерC_3) окажется меньшим y, то переменной y присваивается значение этого элемента (например$y=C_3$). Если значения других элементов массива будут большими C_3, то полученный результат и является

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

наименьшим значением массива. После завершения цикла предусматривается печать значения y, т.е. наименьшего значения элементов массива.

Таким образом, внутри циклических участков алгоритмов могут содержаться также и участки ветвления и следования.

2.6. Описание алгоритмов с вложенными циклами

Вложенным называется цикл, который размещается в теле другого цикла. Телом цикла называют последовательность действий, выполняемых в цикле. На рис.2.14 показан вложенный цикл, который размещается в теле цикла 1. Вложенные циклы напоминают вложенные деревянные "матрешки". В тело одной "матрешки" вкладывается другая "матрешка", в которую, в свою очередь, могут быть помещены еще "матрешки".

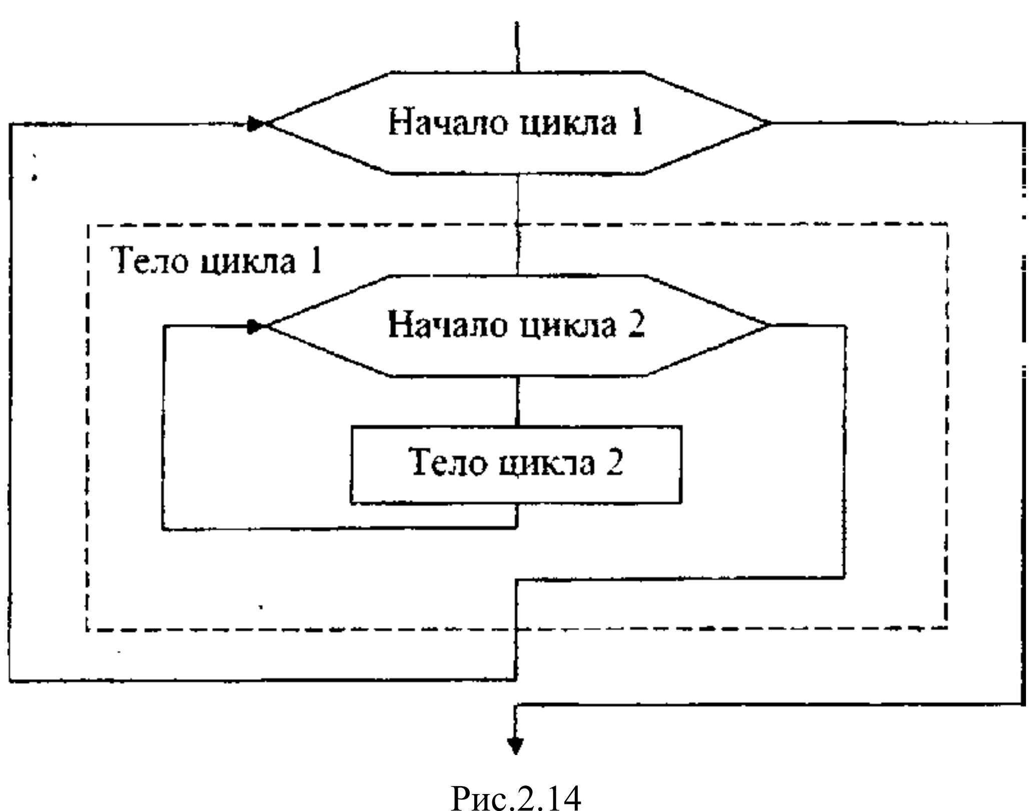


Рис.2.14

Аналогично этому в тело цикла 2 можно вложить цикл 3, а внутри его цикл 4 и т.д. Вложенные циклы обычно применяются при вводе и выводе таблиц данных и таблиц результатов соответственно при табулировании

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

функции по двум и более переменным, при сортировке массивов и при решении многих других задач.

Ввод или вывод таблиц данных сводится к вводу или выводу двумерных массивов.

Элементы двумерного массива обозначаются одной буквой, но с двумя индексами: $b(1,1)$, $b(1,2),...,b(1,j),...,b(1,m)$, $b(2,1),...,b(2,j),...$, $b(2,m),..., b(i,1)b(i,2),..., b(i,j),..., b(i,m),...,b(n,1), b(n,2),...,b(n,j),..., b(n,m)$,

где j — $1,2,..., m$ - номер столбца

$i= 1,2,...,n$ -номер строки.

В дальнейшем материале пособия каждый элемент двумерного массива будет обозначаться переменной с индексами i и j (например, b_{ij}) подобно тому, как это уже применялось при описании одномерных массивов.

Пример 2.8. Построить алгоритм (подалгоритм) ввода в ЭВМ таблицы данных, содержащей m столбцов и n строк.

В этом примере используем процедуру циклического чтения элементов

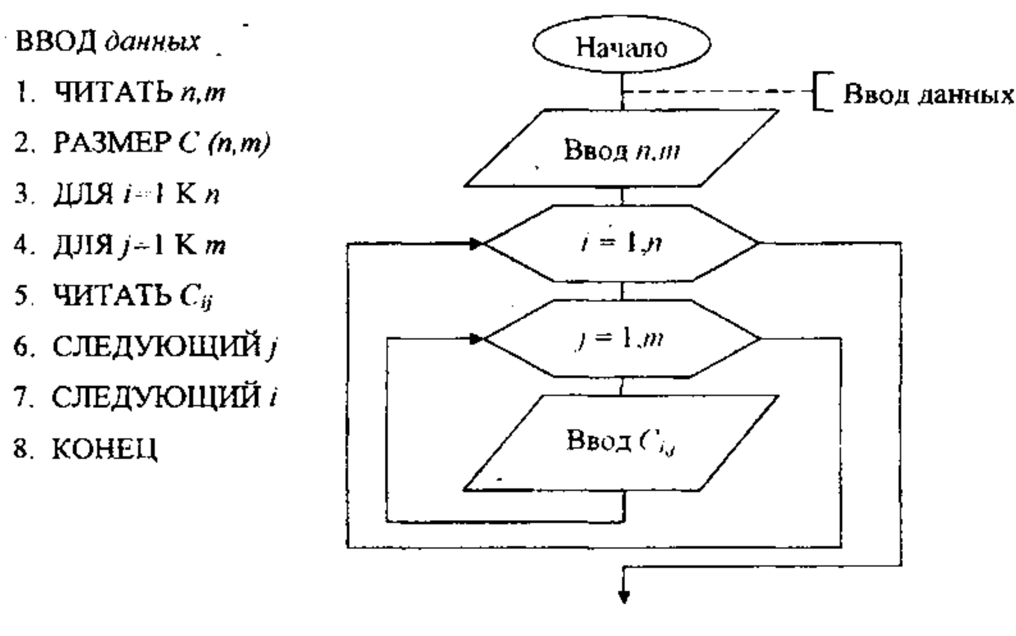


Рис. 2.15

массивов. Она заключается в следующем. Вначале считываются все элемен-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ты первой строки (их число равно m по числу столбцов), затем все элементы второй строки (их число также равно m) и т.д. до считывания m элементов последней n-ой строки. После считывания элементов последней строки таблица данных полностью введена в машину.

На рис.2.15 приведен вариант графической и псевдокодовой записи алгоритма ввода таблицы данных. В этом алгоритме два цикла - внутренний (вложенный) и внешний. Внутренний цикл служит для ввода элементов очередной строки таблицы (ввод по индексу j= 1,…,m) Внешний цикл служит для назначения номера очередной вводимой в компьютер строки.

Задание. Составить алгоритм вычисления суммы элементов матрицы A(5,3).

2.7. Особенности описания алгоритмов, состоящих из нескольких модулей

При записи алгоритмов решения задач часто приходится сталкиваться с ситуацией, когда одна и та же последовательность действий должна выполняться в нескольких разных местах алгоритма. Многоразовая запись одной и той же последовательности действий очевидно будет приводить к увеличению размерности и трудоемкости описания как алгоритма, так и разрабатываемой для реализации его компьютерной программы. Рационально описать эту последовательность действий и соответствующих ей команд программы лишь один раз, предусмотрев обращение к ним по мере необходимости. С этой целью каждую встречающуюся в алгоритме повторяющуюся последовательность действий следует представить в виде отдельного модуля.

В общем случае любой модуль реализует какую-либо функцию по обработке исходных и (или) промежуточных данных и формирует один или несколько выходных параметров.

В качестве модуля может использоваться и некоторый алгоритм

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

представляющий интерес при решении ряда различных задач, например, задач блочной структуры. В этом случае алгоритм программируется и оформляется как программа, а затем используется по мере необходимости для решения различных задач.

Выделение модулей проводят на первых шагах рассмотрения и детализации алгоритма. При этом к модулям целесообразно предъявлять следующие требования.

Во-первых, в качестве модулей следует выделять функционально независимые части общего алгоритма. Это избавит от необходимости вносить изменения в различные модули в случае обнаружения ошибок в одном из них.

Во-вторых, модуль должен иметь один вход и один выход. Это улучшает читаемость алгоритма модуля.

В-третьих, необходимо минимизировать число общих данных, используемых различными модулями. Это увеличивает автономность модулей, упрощает процесс проверки алгоритма.

Разработка схем, состоящих из различных модулей, предполагает описание их функций в виде отдельных схем с последующим использованием этих описаний в схеме другого модуля. Такое использование называют «обращением к модулю» или «вызовом модуля» и в соответствии с этим используют понятия «вызывающий модуль» и «вызываемый модуль».

В программировании данным понятиям соответствуют понятия основной программы и подпрограммы.

Чтобы обратиться к подпрограмме, нужно выполнить следующие действия. Во-первых, нужно задать ее параметры, во-вторых, нужно перейти на начало подпрограммы; в-третьих, нужно вернуться в основную программу. *Параметры подпрограммы* - это те исходные данные, которые необходимы для решения задачи, реализуемой подпрограммой.

В современных языках программирования для организации обращения

к подпрограммам с последующим возвратом в основную программу используется специальный оператор перехода на подпрограмму.

ВЫЗОВ имя подпрограммы (a_1, a_2,...,a_n), *где*a_1, a_2,..., a_n- *входные* и выходные переменные основной программы. Входным переменным присваиваются значения исходных данных, выходным - результаты работы подпрограммы. Через входные и выходные переменные основная программа осуществляет обмен информацией с подпрограммой. Имена входных и выходных переменных обязательно присутствуют как в основной программе, так и в подпрограмме. Указанный оператор перехода вызывает исполнение подпрограммы, которая начинается с оператора.

ПОДПРОГРАММА имя подпрограммы(b_1,b_2,..., b_n)где b_1,b_2,..., b_n- входные и выходные переменные подпрограммы. Все переменные a_1 и b_1 a_2и b_2,..., a_n и b_n должны быть строго согласованы. Пусть, например, a_1, a_2 и a_3- входные переменные из основной программы, а a_4 - выходная переменная. В свою очередь, b_1, b_2 и b_3 - входные переменные, а b_4-выходная переменная подпрограммы. Тогда для очередной реализации подпрограммы нужно переменным b_1,b_2 и b_3 присвоить соответственно численные значения переменных a_1, a_2 и a_3. Полученный результат - это численное значение переменной b_4. Переменной a_4 присваивается этот полученный результат.

Кроме входных и выходных переменных в подпрограмме есть внутренние переменные, которые являются вспомогательными и используются только в подпрограмме. Внутренняя переменная должна обладать тем свойством, что, если везде в данной подпрограмме заменить имя этой переменной на любое другое, не фигурирующее в основной программе, то работа в целом основной программы не изменится.

Для продолжения выполнения основной программы после реализации подпрограммы предусмотрен оператор *ВОЗВРАТ,* который осуществляет

возврат к команде основной программы, следующей за оператором перехода

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

к данной подпрограмме.

При составлении программ часто бывает необходимо обращаться к подпрограммам, которые в свою очередь сами обращаются к другим подпрограммам.

При описании связей между подпрограммами следует избегать обращения подпрограммы самой к себе как непосредственно, так и через другие подпрограммы.

Описание подпрограмм в алгоритмах следует вести ориентируясь на последующие возможности их представления в программе, исходя из того, что предоставляет для этого выбранный язык программирования. Например, в языке программировакия«Паскаль»подпрограммы могут описываться в виде функций или процедур.

Возможно использование функций двух видов - встроенных (стандартных) и создаваемых программистом. В любом случае функция определяет одно значение, которое передается через ее имя.

Встроенные функции являются неотъемлемой частью языка программирования, т. е. их имена могут использоваться только по прямому назначению. Назовем для примера имена нескольких встроенных функций языка Паскаль: abs, exp, In, sqr, sqrt. Набор встроенных функций Паскаль весьма обширен. Поэтому его полезно изучить, прежде чем приступать к описанию алгоритма. И только в случае невозможности использовать стандартную функцию разработчик алгоритма должен предусмотреть создание программистом собственной функции. Такая функция в отличие от встроенной будет описываться в разделе описания программы. Создаваемая программистом функция начинается со служебного слова function, после которого следует выбранное программистом имя функции. Подобным образом с использованием служебного слова procedure описываются процедуры. Обращения и к созданной функции и к процедуре осуществляются по их имени.

Глава 3. Вычислительные алгоритмы инженерных задач

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

3.1. Общие подходы к разработке алгоритмов

Изучением изложенного в этой главе материала могут достигаться две цели. Во-первых, это изучение типовых приемов разработки алгоритмов. Выдающийся математик и педагог Д. Пойа считал, что решение задач - практическое искусство, подобное плаванию, катанию на лыжах или игре на фортепиано, научиться которым можно, только подражая хорошим образцам и постоянно практикуясь. Во-вторых, возможность использования приведенных ниже конструкций алгоритмов при необходимости решения соответствующих задач.

Известно, что для большинства инженерных задач может быть использовано несколько алгоритмов, приводящих к получению решения. Применение различных алгоритмов может потребовать различных затрат времени на их реализацию, различной организации работ исполнителя (например, задание значений исходных данных одним массивом или пошагово в процессе решения). Выбор того или иного алгоритма часто определяется конкретным видом математической модели. При выборе алгоритма ориентируются также на его исполнителя (человека, ЭВМ, автоматическое устройство управления). В любом случае при выборе алгоритма необходимо оценивать его способность обеспечить достижение решения для всех определенных в постановке задачи условий.

Рассмотрение подходов к разработке алгоритмов начнем применительно к последнему положению, обратившись с этой целью к приведенному в разделе (п. 1.1) алгоритму решения квадратного уравнения, который был оценен как некорректный, так как в нем не учитывались все возможные задаваемые значения исходных данных. Очевидно, что для того, чтобы построить универсальный алгоритм, предварительно требуется тщательно проанализировать математическое содержание задачи. Выполним эту работу применительно к рассматриваемому примеру.

Получение решения уравнения зависит не только от значения дискриминанта *d,* но и от значений коэффициентов *a, b*, *c.* Исходя из этого, проведем анализ возможных вариантов решений, ограничиваясь только поиском вещественных корней. При условии, что:

$a \neq 0$ *и* $d \geq 0$ - уравнение имеет два вещественных корня;

$a=0,\ b \neq 0$ - уравнение линейно и имеет одно решение:$x = -c/b;$

$a= 0, b=0, c=0$- любое x будет решением уравнения;

$a \neq 0$ *и*$d<0$- уравнение не имеет вещественных корней;

$a=0, b=0, c \neq 0$ - уравнение не имеет решений.

В соответствии с выполненным анализом блок-схема алгоритма решения рассматриваемой задачи может иметь вид, представленный на рис. 3.1.

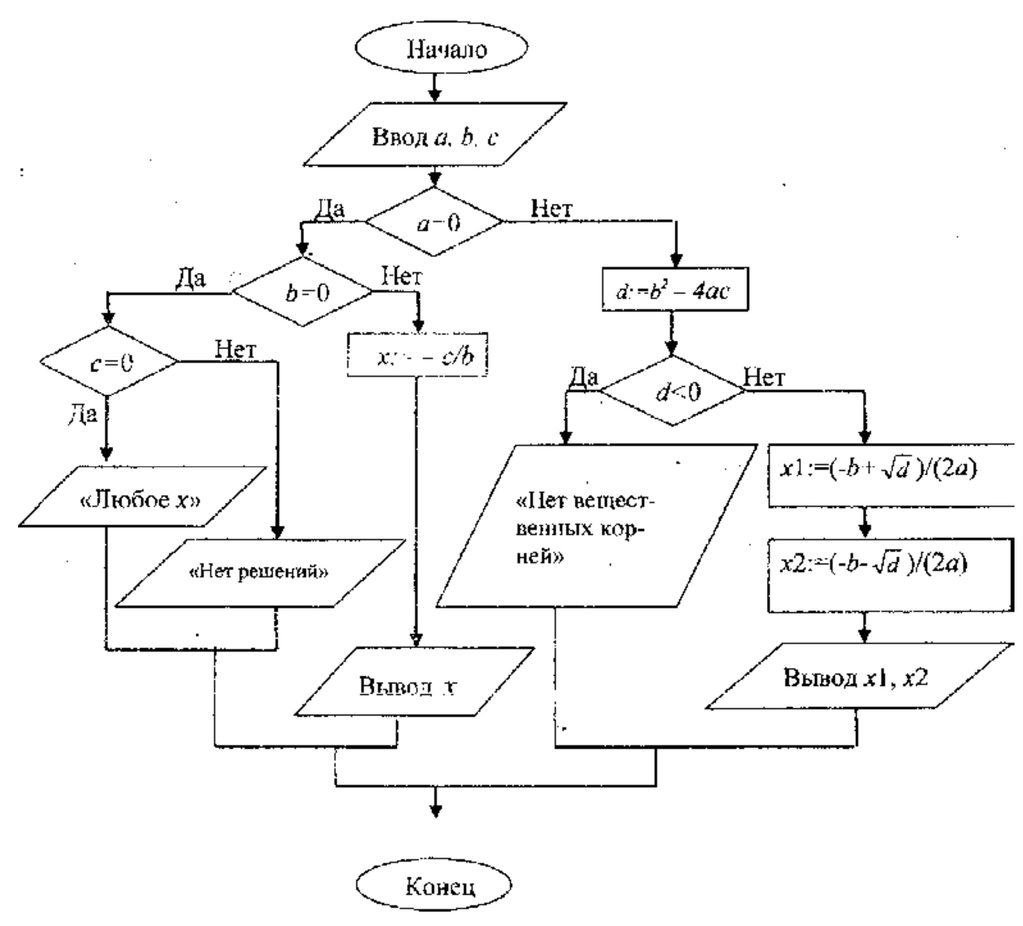


Рис. 3.1

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Рассмотрим еще один возможный подход к разработке алгоритма решения квадратного уравнения. К нему можно подойти с такой позиции: еслиa=0, то это уже не квадратное уравнение й его можно не рассматривать. В таком случае будем считать, что пользователь ошибся при вводе данных, и следует предложить ему повторить ввод. Иначе говоря, в алгоритме будет предусмотрен контроль достоверности исходных данных с предоставлением пользователю возможности исправить ошибку. Наличие такого контроля - признак хорошего качества алгоритма. Блок-схема этого алгоритма представлена на рис. 3.2.

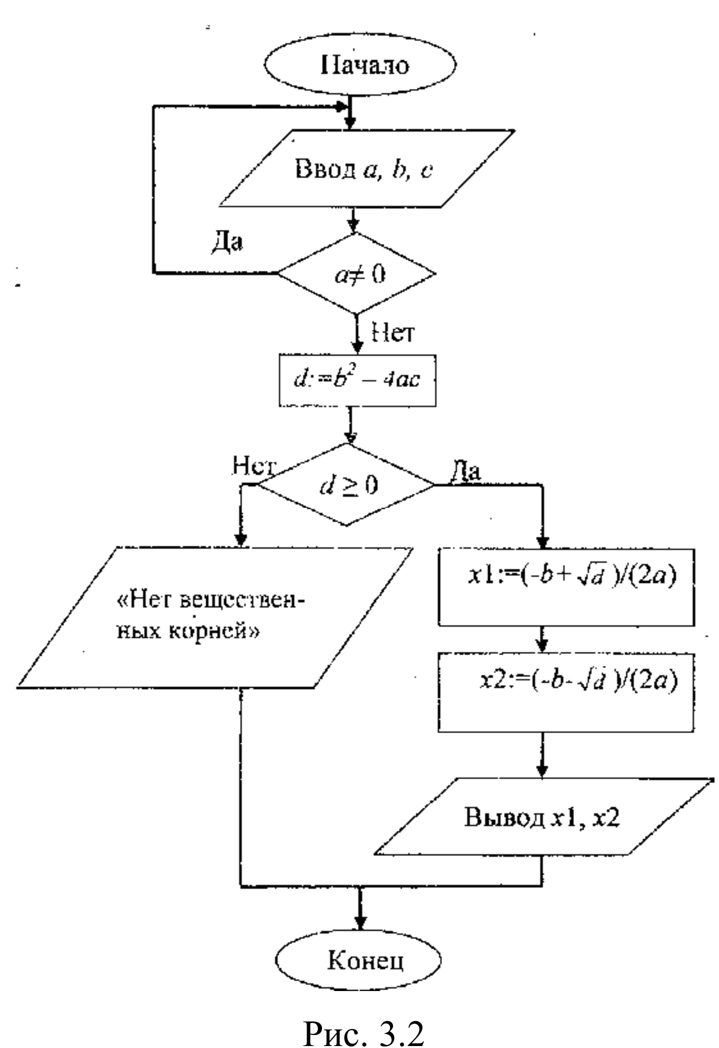


Рис. 3.2

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

3.2. Последовательное суммирование и последовательное умножение

Освоение предлагаемых ниже алгоритмов чрезвычайно важно в методическом отношении для обеспечения разработки эффективных алгоритмов решения на ЭВМ большого количества разнообразных задач.

Пример 3.1. Алгоритм вычисления суммы чисел: пусть требуется вычислить сумму

$$S = \sum_{i=1}^{n} f(x_i),$$

при известной функции *f(x)* и заданных значениях аргумента *x*. В простейшем случае это может быть массив числовых значений исходных данных, читаемых из входного файла.

Возможны, по крайней мере, два подхода к решению этой задачи.

Первый подход заключается в том, что вначале вычисляются все *n* значений функции

$$y_i\text{-}\ f(x_i),$$

каждое из которых запоминается отдельно в памяти компьютера. Затем все значения суммируются, т.е.

$$S = \sum_{i=1}^{n} y_i$$

Как видно, данный подход потребует использования значительного объема компьютерной памяти.

Второй подход более рационален. Он реализуется накоплением промежуточной суммы, что обеспечивается за счет введения в алгоритм *накатывающего счетчика.* Суть алгоритма при этом состоит в следующем. Вначале производится исходная установка счетчика заданием операции присваивания $S = 0$, которая носит название задания начального условия. Затем производится вычисление очередного значения y_i (на первом шаге для $i = 1$) и его суммирование с значением, находящимся в счетчике (на первом шаге $S = 0$).

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Процесс последовательного суммирования может быть представлен в виде последовательности следующих операций:

$$S_0=0;$$

$$S_1=y_1=S_0+y_1;$$

$$S_2=S_1+y_2;$$

$$\dots$$

$$S_n=S_{n-1}+y_1.$$

На основе этих выражений формулируется следующий оператор присваивания.

$$S = S + y_1.$$

Графическая запись этого алгоритма приведена на рис. 3.3.

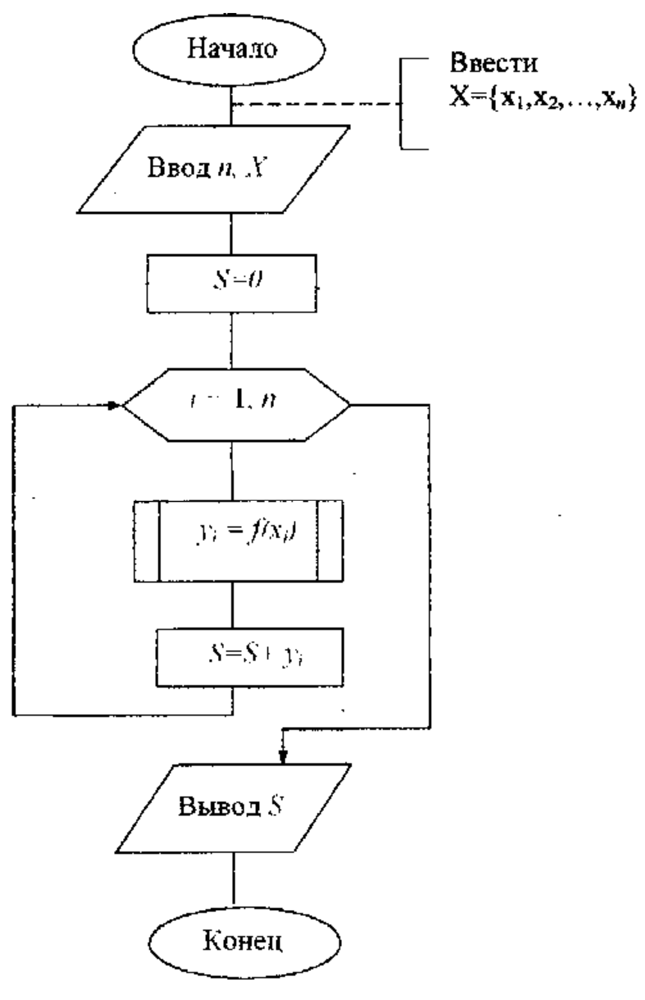


Рис. 3.3

Достоинством второго подхода является то, что здесь ограничивается использование памяти компьютера для хранения промежуточной информации без увеличения практически времени на производство вычислений. Рассмотрение приведенного алгоритма позволяет сделать следующий вывод: при суммировании нескольких чисел целесообразно задавать начальное условие S=0.

Пример 3.2. Требуется найти произведение *i* вычисляемых значений функции *f(x)*, т.е.

$$P = \prod_{i=1}^{n} f(x_i)$$

при известной функции *f(x)* и заданных значениях аргумента *x*.

Так же как и при решении предыдущей задачи, возможны два подхода. Первый подход состоит в предварительном вычислении всех *n* значений

$$y_i=f(x_i)$$

и в последующем их перемножении.

Рассмотрим второй подход, для которого определим:

$$P_0=1;$$

$$P_1=y_1=P_0 \cdot y_1;$$

$$P_2=P_1\ y_2,$$

$$\ldots$$

$$P_n=P_{n-1}.y_n.$$

В общем виде оператор присваивания имеет вид:

$$P = P.y_i.$$

В данной задаче начальным условием является P = 1.

Графическая запись алгоритма приведена на рис. 3.4.

Данный алгоритм отличается от ранее рассмотренного, главным образом, начальным условием. При решении этой задачи обнаружено очень важное обстоятельство: при перемножении нескольких сомножителей целесообразно задавать начальное условие $P=1$.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Рассмотренные подходы находят применение при решении на ЭВМ многих задач, в частности, ряда рассматриваемых ниже примеров.

Рис.3.4

3.3. Понятие итерационных алгоритмов, их описание

Термин *итерация* произошёл от латинского слова iteratio - *повторение.* Итерационным процессом задается последовательное выполнение однотипных операций. Этот процесс представляет собой движение вперед - от предыдущего результата вычислений к последующему. При этом все вычисления обязательно выполняются с помощью одной и той же математической операции. Так, если

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

$$y = f(x) \equiv f_1(x)$$

есть некоторая функция от ***x,*** то функции

$$f_2(x)=f[f_1(x)],\ f_3(x)=f[f_2(x)],\ \dots,\ f_n(x)=f[f_{n-1}(x)]$$

называются соответственно второй, третьей, ..., *n*-й итерациями функции $f(x)$. Например, для

$$f(x) = f_1(x) = (ax)^{\beta},$$

следует

$$f_2(x) = ((ax)^{\beta})^{\beta} = (ax)^{\beta 2},\ f3(x) = (ax)^{\beta 3},\dots,\ f_n(x) = (ax)^{\beta n}.$$

Индекс *n* называют показателем итерации, а переход от функции $f_{i-1}(x)$ к функции $f_i(x)$- шагом итерации. Очевидно, что показатель итерации совпадает с количеством шагов итерационного процесса.

Неотъемлемой частью итерационного процесса является накапливающий счетчик числа итераций, реализуемый либо с помощью операции присваивания $i= i+1$, либо с помощью операторов цикла ДЛЯ i=1 *к n* и СЛЕДУЮЩИЙ.

Пример 3.3. Требуется вычислить $p = n!$ Известно, что $n! = 1 * 2 * 3 \dots i \dots n$, где i- целое число. В этом примере:

$$p_0 = 1, p_1 = f(P_0) = p_0 *1,\ P_2 = f(p_1) = p_1 *2,$$

$$P_3 = f(P_2) = P_2 *3,\dots,p_{i+1} = f(P_i) = P_i *i, P_n = f(P_{n-1}) *n.$$

Таким образом, начальное условие здесь *p*-1, а шаг итерации (переход от функции P_iк функции p_{i+1}) реализуется с помощью выражения $p=p*1$.

На рис.3.5.а и 3.5б приведены две графические записи итерационного алгоритма решения рассматриваемого примера для двух способов реализации счетчика числа итераций соответственно:

Напомним, что по определению при *n=0 и n!=1.* Очевидно, что для возникновения такой ситуации в приведенные выше блок-схемы алгоритма должны быть включены дополнительные операторы. Обучаемым рекомендуется самостоятельно выполнить эту работу.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

а б

Рис.3.5

3.3.1. Итерационные алгоритмы с заданным числом итераций

Итерационные алгоритмы подразделяются на две группы: с заданным числом итераций, с вычисляемым в ходе итерационного процесса числом итераций. Первая группа алгоритмов относится к классу алгоритмов точных вычислений, вторая - к классу приближенных вычислений. Строго говоря, ни один из реализуемых на ЭВМ алгоритмов не обеспечивает точные вычисления вследствие инструментальной погрешности, обусловленной ограниченной разрядной сеткой ЭВМ. Однако в алгоритмах приближенных вычислений к инструментальной погрешности добавляется методическая погреш-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ность, связанная со способом определения количества итераций и, следовательно, завершения итерационного процесса.

Простейшие итерационные алгоритмы первой группы рассматриваются в данном разделе. Эти алгоритмы широко применяются в задачах вычисления многочленов и обработки результатов измерений.

Пример 3.4. Пусть требуется вычислить многочлен

$$y = a_3x^3 + a_2x^2 + a_1x + a_0,$$

где x, a_0, $a_ь a_ъ$ a_3 - действительные числа.

Решение этой задачи “в лоб”, без привлечения-упрощающих схем приводит к избыточным вычислениям. Так, в рассматриваемом примере потребуется выполнение шести длинных операций умножения и трех операций сложения:

$$y - a_3 * x * x * x + a_2 * x * x + a, * x + a_0.$$

Преобразуем данный многочлен следующим образом:

$$y = ((a_3x + a_2)x + a_{n-2})x + \dots +)x+a_0.$$

Теперь для вычисления многочлена достаточно выполнить только три операции умножения и три операции сложения. Для произвольной степени nмногочлена управляющая схема вычислений следующая:

$$y= (((a_nx+a_{n-1})x + a_{n-2})x + \dots + a_1)x+a_0.$$

Эта схема вычислений известна как схема Горнера. Она требует только nопераций умножения и n операций сложения, тогда как обычная схема вычислений требует $n(n+1)/2$ операций умножения и n операций сложения. Например, для вычисления многочлена степени n=7 по схеме Горнера достаточно ограничиться только семью операциями умножения, тогда как обычная схема потребует их увеличения до 21.

В целях обеспечения удобства построения блок-схемы итерационного алгоритма переименуем одномерный массив действительных чисел

$$\{a_n,a_{n-1},\dots,a_i,\dots,a_1,a_0\}=\{b_0,b_1\dots,b_i,b_{n-1},b_n\}$$

С учетом новых обозначений исходный многочлен имеет вид

$$y = b_0x^n + b_1x^{n-1} + \ldots + b_1x^{n-1} + \ldots + b_{n-1}x + b_n,$$

а схема вычислений Горнера:

$$y = (((b_0x + b_1)x + b_2)x + \ldots + b_{n-1})x + b_n.$$

Примем $y_o = b_0$. Тогда

$$y_1 = y_0x {}^{+}b_1, y_г = y_1x + b_2, y_3 = y_2x + b_3, \ldots, y_n = y_{n-1}x + b_n.$$

Соответственно, оператор присваивания будет иметь вид:

y-yx-b_i,гдеi = 1,2,...,n.

На рис.3.6 приведена графическая запись итерационного алгоритма с числом итераций *n* для вычисления многочлена степени *n* по схеме Горнера.

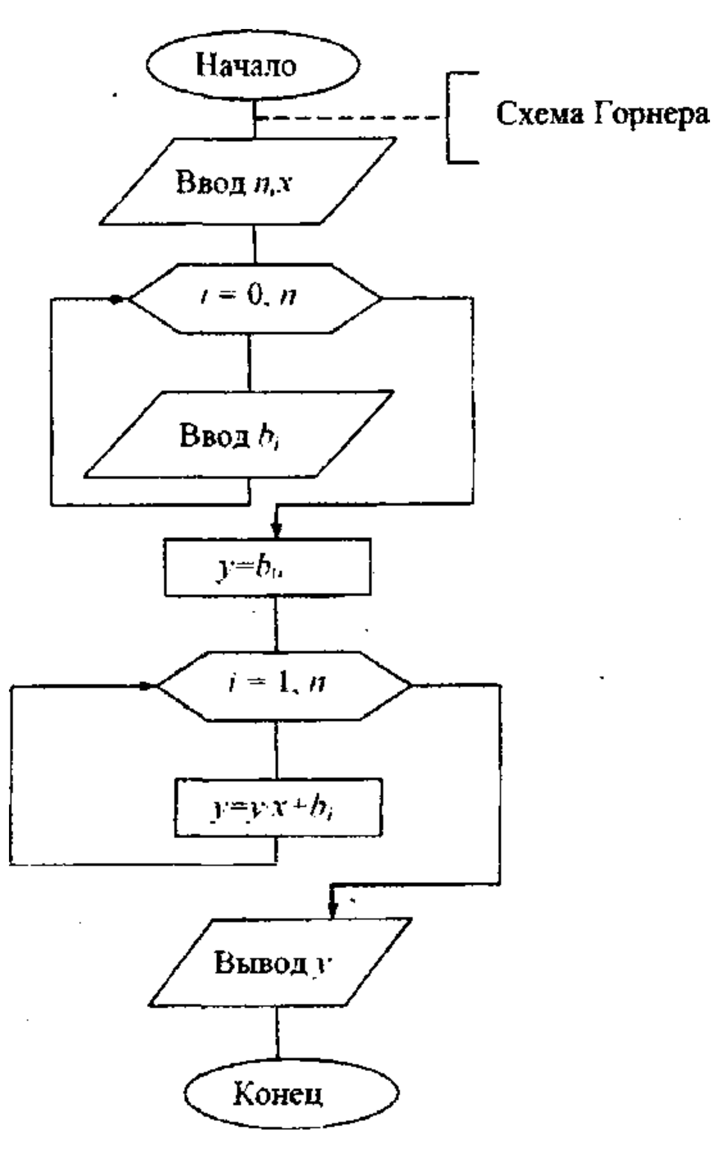


Рис. 3.6

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Пример 3.5. Рассмотрим пример построения итерационного алгоритма для интерполирования функций. Пусть относительно исходных значений x_1, $x_2,..., x_n$ получены в результате эксперимента выходные параметры $y_1, y_2...$, y_nработы какого-либо объекта. По этим n данным требуется построить функцию $y = f(x)$, что позволит вычислять промежуточные значения $_y_i$ при любом заданном значении аргумента x_i. Для определения значения функции $f(x)$ строят функцию $F_n(x)$, которая равна

$$F_n(x_1) = y_1,\ F_n(x_2) = y_2, ...,\ F_n(x_n) = y_n,$$

а в остальных точках отрезка $(x_1,\ x_n)$ представляет функцию $f(x)$ приближенно. Задача построения функции

$$F_n(x_1) = f(x_1), ..., F_n(x_n) = f(x_n)$$

называется задачей интерполяции.

Чаще всего функцию $F(x)$ отыскивают в виде полинома степени n, используя интерполяционную формулу Лагранжа

$$F_n(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j \neq i}} \frac{x - xj}{xi - xj}$$

где x - промежуточное значение аргумента.

Схема алгоритма интерполяции функции методом Лагранжа приведена на рис.3.7. Он содержит операции суммирования переменных по индексу j. С этой целью введены начальные условия FN=0 (для реализации операции суммирования) и P= 1 (для реализации операции умножения).

Первые три блока обеспечивают ввод исходных данных. Ими являются массивы X и У. Затем задаются начальные условия $FN=0$ и P=1 соответственно суммирования и определения произведений. Далее организуют циклы и осуществляют соответственно вычисления произведений и сумм. С помощью операции $j \neq i$ исключается возможность возникновения в знаменателе формулы Лагранжа разности одинаковых элементов, что могло бы привести к делению на 0.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

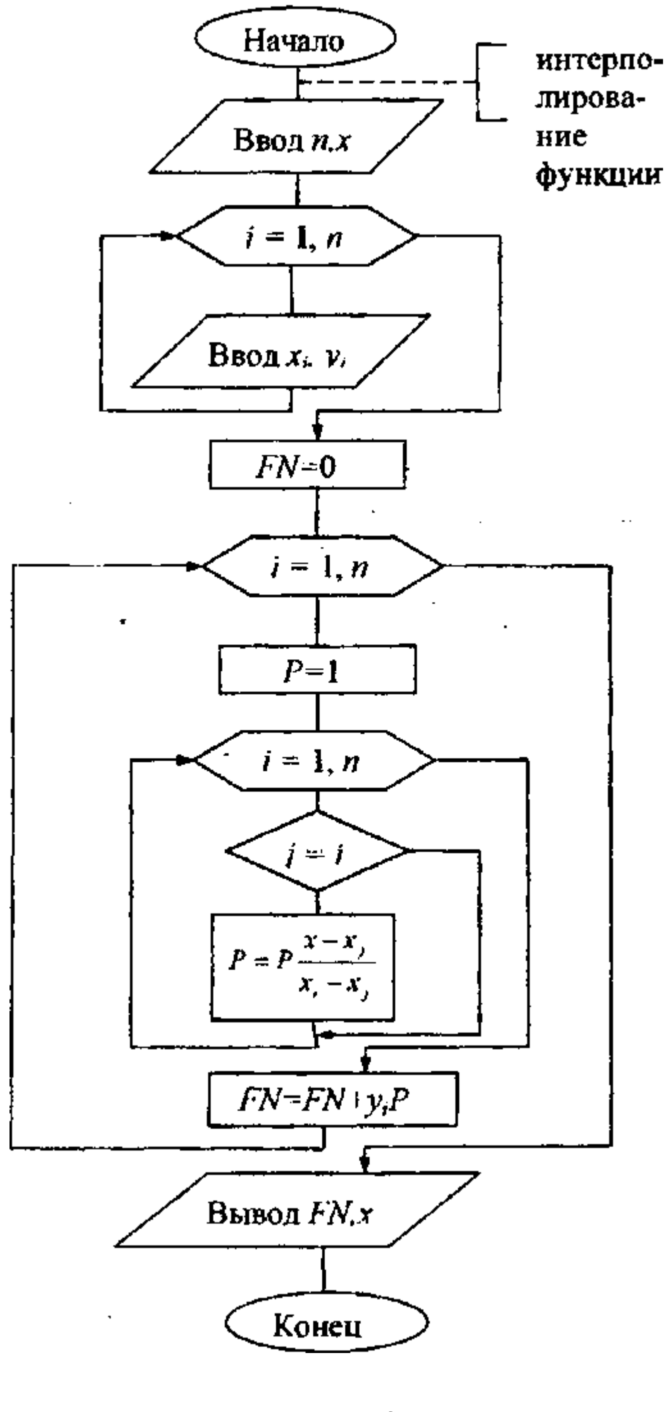


Рис. 3.7

Пример 3.6. Для уменьшения ошибок измерений параметров каких-либо физических величин применяют "сглаживание" данных по определенным формулам, в которых усредняются результаты a_{i-1}, a_i, a_{i+1} смежных измерений. Пусть определены действительные данные $a_1,\ldots,a_n$ требуется получить "сглаженные" значения $b_1,\ldots, b_m$ заменив в исходной последовательности все

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

члены, кроме первого и последнего, по формуле

$$b_i = \frac{b_{i-1} + a_i + a_{i+1}}{3}$$

Задача решается с помощью итерационного алгоритма, содержащего *n-2* итераций, поскольку по условию $b_1 = a_1$ и $b_n = a_n$. Из этого условия также следует, что i=2,3,..., *n-1*.

Блок-схема итерационного алгоритма решения рассматриваемой задачи приведена на рис.3.8.

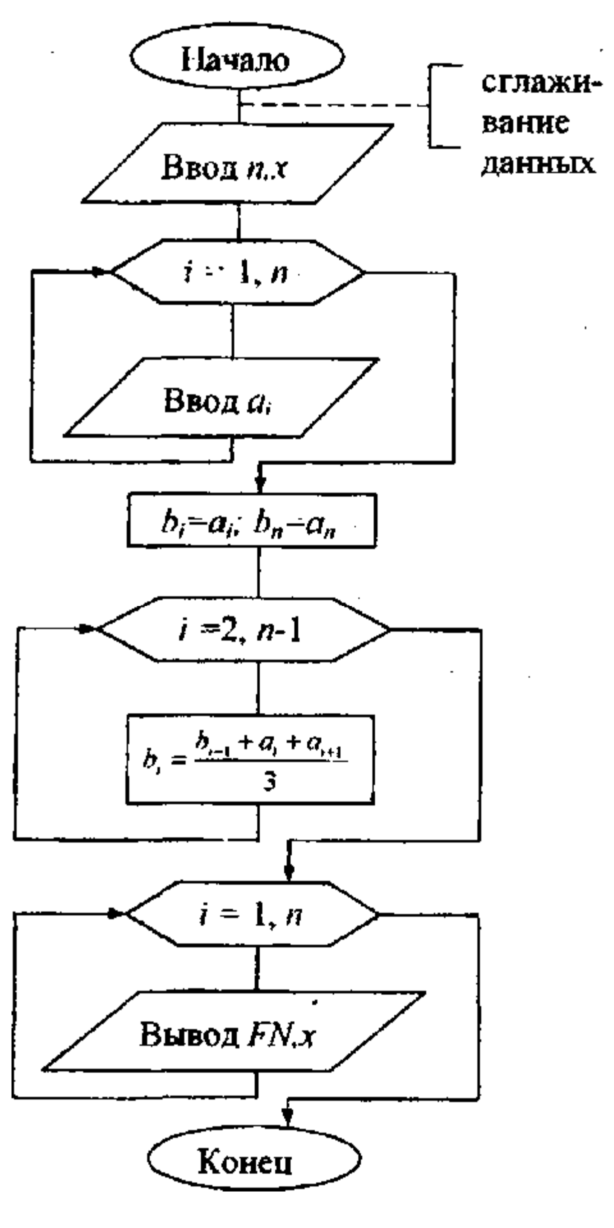


Рис. 3.8

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

3.3.2. Итерационные алгоритмы последовательных приближений

При решении многих задач вычислительной математики количество итераций предварительно неизвестно, а задано только требование по точности вычислений. По этой причине на каждом шаге работы программы необходимо анализировать соответствие достигнутой точности заданному требованию и в зависимости от этого либо прекращать итерационный процесс, либо выполнять очередную итерацию.

Шаг вычислений будем называть нерегулируемым, если в ходе всего итерационного процесса не изменяются начальные условия задания аргументов вычисляемой функции. Он может быть постоянным в тех случаях, когда приращение аргумента постоянно, т.е.

$$x_n=x_{n-1}+\Delta x,$$

где Δx=const, либо переменным, если

$$x_n-x_{n-1}=\text{var}.$$

Шаг вычислений является регулируемым, если в ходе итерационного процесса автоматически корректируются предыдущие исходные данные для выбора шага. Например, после n итераций приращение аргумента уменьшается в $к$ раз, т.е.

$$x_{n+1}=x_n+\Delta x/k.$$

Итерационные алгоритмы с регулируемым шагом вычислений принято называть *алгоритмами с автоматическим выбором шага.*

Изображение структуры итерационных алгоритмов с нерегулируемым шагом вычислений, заданной точностью ε и достигнутой по завершении очередной итерации абсолютной величиной точности |R| может быть таким, как это изображено на рис. 3.9. Из представленной схемы следует, что процесс вычислений будет продолжаться до тех пор, пока вычисления значения величины |R| превышает значение заданной точности ε.

Задание исходных данных

Задание начальных условий

Вычисление результатов очередной итерации

|R|>ε

Нет

Да

Завершение итерационного процесса

Рис.3.9

Изображение структуры итерационных алгоритмов с автоматическим вы бором шага может быть описано:

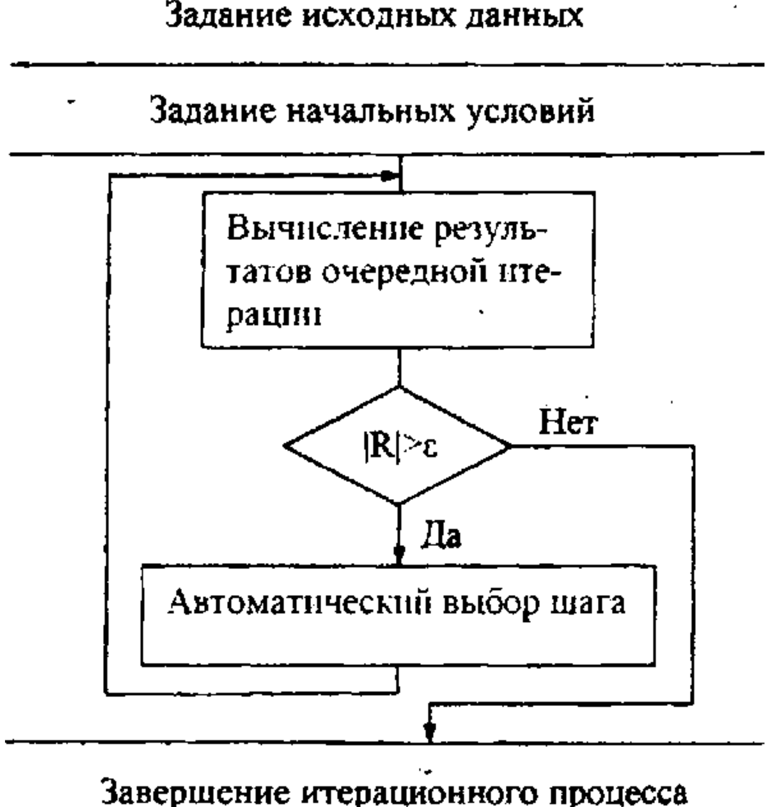


Рис. 3.10

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Итерационный процесс с постоянным или переменным шагом и с вычисляемым количеством итераций принято называть процессом последовательных приближений.

Данная группа алгоритмов применяется при вычислении бесконечных сходящихся рядов, при решениях алгебраических и трансцендентных уравнений, систем алгебраических и дифференциальных уравнений, при вычислении интегралов и т.п.

Рассмотрим характерные примеры алгоритмизации вычислительных задач с помощью процесса последовательных приближений.

Пример 3.7. Составить схему алгоритма вычисления суммы сходящегося ряда

$$S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

с точностью ε.

Поскольку ряд сходится для любых значений *x,* достаточным условием обеспечения заданной точности является достижение очередным членом ряда $h_n = \frac{x^n}{n!}$ $|h_n| \le \varepsilon$, так как при этом следующий член ряда заведомо не превысит величины ε.

Для определения значений членов ряда воспользуемся соотношением

$$h_{n+1} = h_n x/(n+1),$$

поскольку прямое вычисление $(n+1)$-го члена ряда по формуле $x^{n+1}/(n+1)!$ требует большого времени.

Так как $h_0 = 1$, то достаточно взять это значение в качестве исходной переменной *h*. Суммирование вычисляемых последовательно членов ряда удобно производить, используя способ накопления суммы. В нашем примере начальное значение суммы целесообразно считать равным не нулю, а $S = h_0 = 1$. Схема алгоритма решения задачи представлена на рис.3.11.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

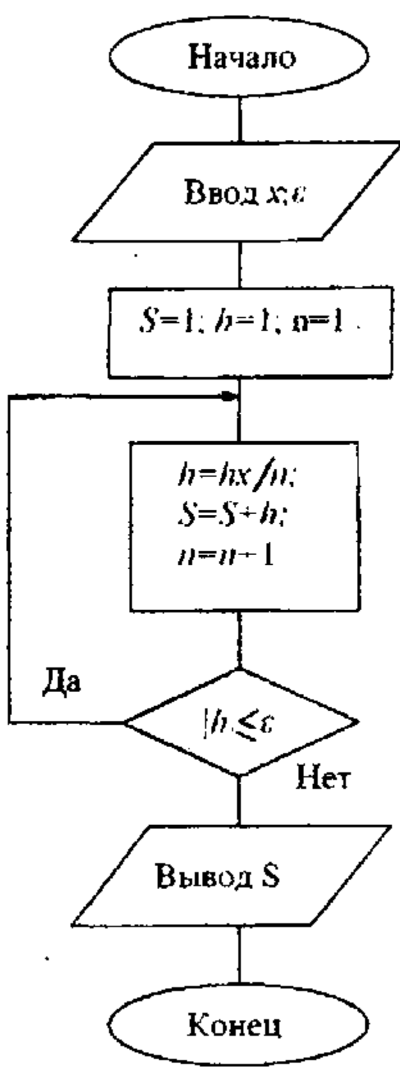


Рис.3.11

Первый блок обеспечивает задание исходных данных, второй блок - начальных условий, затем вычисляются результаты очередной итерации и принимается решение о продолжении или прекращении итерационного процесса.

В рассмотренном примере количество итераций вычисляется в зависимости от достигнутой точности по модулю, шаг вычислений не регулируется.

Пример 3.8. Составить схему алгоритма решения алгебраических и трансцендентных уравнений методом итераций.

Решение этой задачи сводится к нахождению корней уравнения вида $f(x)=0$, где функция $f(x)$ определена и непрерывна на некотором интервале. Если функция $f(x)$ - многочлен, то данное уравнение называется алгебраическим; если же в функцию $f(x)$ входят трансцендентные (тригонометрические,

логарифмические, показательные и т.п.) функции, то уравнение $f(x)=0$ называется трансцендентным.

Задача выполняется в два этапа: 1) определение корней, т.е. отыскание достаточно малых отрезков на оси *x,* в каждом из которых заключен один и только один корень уравнения; 2) уточнение корня с заданной точностью.

Предположим, что известен отрезок $[a,b]$ на оси *x,* внутри которого существует один и только один из корней уравнения $f(x)=0$. Представим это уравнение в виде $x=x+\lambda f(x)$, где $\lambda\neq0$ является произвольной константой. Обозначим правую часть $x+\lambda f(x)=\psi(x)$ и, следовательно, $x=\psi(x)$. Выберем теперь на отрезке $[a,b]$ произвольную точку x_0 и последовательно будем вычислять

$$x_1=\psi(x_0)$$

$$x_2=\psi(x_1)$$

$$\dots$$

$$x_n=\psi(x_{n-1})$$

Процесс последовательных вычислений значений $x_1, x_2,\dots,x_n$по приведенным формулам является итерационным процессом.

Этот процесс сходится к корню уравнения $x=\psi(x)$*,* если на отрезке $[a,b]$выполнено условие$|\psi'(x)|\leq q<1$, которое часто можно обеспечить за счет удачного подбора коэффициента λ.

Чтобы обеспечить процесс вычислений с точностью ε, процесс итераций следует продолжать до тех пор, пока для двух последовательных приближений x_n и x_{n-1} будет выполняться условие:

$$|x_n - x_{n-1}| \leq \varepsilon_1 = \frac{1-q}{q}\varepsilon$$

при этом всегда будет выполнено

$$|x^* - x_n| \leq \varepsilon$$

где ε- заданная предельная абсолютная погрешность истинного значения корня x^*.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Приведенные соображения позволяют представить схему искомого алгоритма в виде, показанном на рис.3.12.

Рис. 3.12

В первом блоке задаются исходные данные, во втором - начальные условия, в третьем блоке - вычисляется приближенное значение корня в очередной итерации, в четвертом - вычисляется абсолютная величина отклонения между результатами текущей и предыдущей итерации, затем принимается решение о продолжении или прекращении итерационного процесса и, при необходимости, подготавливаются исходные данные для очередной итерации. В данном алгоритме не регулируется шаг вычислений.

Пример 3.9. Составить схему алгоритма вычисления определенных интегралов методом трапеций.

Решение этой задачи сводится к приближенному вычислению площади, ограниченной сверху кривой подынтегральной функции $f(x)$, а слева и справа прямыми линиями, проведенными параллельно оси ординат и пересекающи-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ми функцию $f(x)$ в точках$f(a)$ иf(b), где $[a,b]$ - отрезок существования аргумента x.

Задача выполняется следующим образом. Отрезок разделяется на n равных частей $h=(b-a)/n$. Затем проводятся n+1 прямых линий, параллельных оси ординат, до пересечения с функцией $f(x)$ и строится n трапеций. Вершины первой трапеции находятся в точках $f(a)$ и $f(a+h)$, а ее площадь

$$T_{1h} = h\left[\frac{f(a)+f(a+h)}{2}\right].$$

Площадь i-й трапеции

$$T_{ih} = h\left[\frac{f(a+(i-1)h)+f(a+ih)}{2}\right],$$

а площадь n-й трапеции

$$T_{nh} = h\left[\frac{f(a+(n-1)h)+f(b)}{2}\right].$$

Очевидно, что приближенное значение интеграла при шаге вычислений h равно

$$T_h = \sum_{i=1}^{n} T_{ih} = h\left[\frac{f(a)+f(\mathrm{b})}{2}+\sum_{i=1}^{n-1} f(x_i)\right],$$

где $x_i=a+ih$.

При этом имеет место абсолютная погрешность вычислений $R_h=J-T_h$, где J - истинное значение интеграла.

Если изменить начальное условие и разделить отрезок $[a, b]$ не на n, а на $2n$ равных частей $h/2=(b-a)/2n$, то приближенное значение интеграла $T_{h/2}$ приблизится к истинному значению и будет отличаться от него величиной абсолютной погрешности $R_{h/2}=J-T_{h/2}$ которая в 4раза меньше погрешности R_h.

Следовательно, $T_h+R_h=T_{h/2}+R_{h/2}$ или $\mathrm{T_h}+4R_{h/2}=T_{h/2}+R_{h/2}$- Отсюда

$$R_{\frac{h}{2}} = \frac{T_{\frac{h}{2}} - T_h}{3}$$

Процесс изменения исходного значения для выбора тага вычислений можно продолжать до тех пор, пока будет выполнено условие

$$\frac{T_{\frac{h}{2}} - T_h}{3} \le \varepsilon ,$$

где ε - заданная абсолютная погрешность вычисления интеграла.

Схема алгоритма решения задачи показана на рис.3.13.

Рис.3.13

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Первый блок обеспечивает ввод данных. Во втором блоке присваивается начальное значение интеграла. В третьем блоке определяется величина *h* шага вычислений и начальное значение суммы высот трапеций. Затем вычисляются текущие значения аргумента *x* и суммы высот трапеций. Далее рассчитываются интеграл *TN2* в данной итерации и абсолютная погрешность *R* этого расчета.

Если абсолютная погрешность *R* больше заданной, то изменяется шаг вычислений - он уменьшается в два раза за счет увеличения в два раза числа *n*частей на интервале [*a,b*]. Таким образом, в рассмотренном алгоритме производится автоматический выбор шага в течение ряда итераций.

3.4. Алгоритмы обработки массивов

Рассмотрим типовые приемы работы с массивами, а также общие подходы к организации алгоритмов обработки массивов.

1. *Удаление элемента из одномерного массива.* Требуется удалить *l*-й элемент из массива*A* размера n. Удалить элемент, расположенный на *l*-м месте в массиве, можно, сдвинув весь “хвост” массива, начиная с (*l*+1)-го элемента, на одну позицию влево, т.е. выполняя операции $a_i=a_{i+j}$, *l+1*,...,
n -1. Размер массива уменьшается на 1. Эта задача реализуется спомощью следующих предписаний:

ДЛЯ*i=l*К*n*

A(i)=A(i+1)

СЛЕДУЮЩИЙ*i*

n=*n*-1.

2. *Включение элемента в заданную позицию одномерного массива.* Перед включением элемента в *l*-ю позицию необходимо раздвинуть массив, т.е. передвинуть “хвост” массива на одну позицию вправо, выполняя операцию $a_{i+1}=a_i$, *i=n, n*-1,…, *l*. Перемещение элементов массива нужно начинать с

“хвоста” с шагом -1 (минус один шаг), т.е. шагом, направленным в сторону

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

уменьшения номеров элементов. При достижении l-ого элемента, ему присваивается заданное значение B. Размер массива увеличивается на 1. Эта задача реализуется с помощью следующих предписаний:

ДЛЯi=nКl- 1

$A(i+1)=A(i)$

СЛЕДУЮЩИЙi

$A(i)=B$

n=n+1.

3. *Удаление строки из матрицы размера (n*x *m).* Требуется удалить строку с заданным номером l. Решение задачи аналогично удалению элемента из одномерного массива. Все строки, начиная с (l+1)-й, нужно переместить вверх. Число строк уменьшается на 1. Эта задача реализуется с помощью следующих предписаний:

ДЛЯ i=l К n

ДЛЯj=1 Кm

СЛЕДУЮЩИЙj

СЛЕДУЮЩИЙ i

n=n-1.

Удаление столбца осуществляется аналогично, но по переменным m и j.

4. *Включение строки в матрицу (n xm).* Включаемая строка задается как вектор С. Ее включение производится аналогично включению элемента в одномерный массив. При этом перемещаются строки, начиная с l-й, вниз (в обратном порядке). Перемещение одной строки связано с пересылкой всех элементов этой строки, что требует организации цикла по переменной j (по номеру столбца):

ДЛЯ i=n Кl ШАГ -1

ДЛЯj=1КM

A(i+1, j)=A(ij)

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

СЛЕДУЮЩИЙj

СЛЕДУЮЩИЙ *i*

ДЛЯj=1К*M*

A(*lj*)=C(j)

СЛЕДУЮЩИЙ *j*

n=*n*+ 1.

Включение столбца осуществляется аналогично.

5. *Перестановка элементов в одномерном массиве.* Требуется переставить *l*-й и (*i*+*k*)-й элементы массива. С этой целью вводится вспомогательная переменная *R,* в которую временно помещается один из переставляемых элементов:

R=A(*i*+*k*)

A(*i*+*k*)=A(*i*)

A(*i*)=*R*.

6. *Перестановка строк матрицы.* Требуется переставить *A(ij)* и *A(i+kj)* строки матрицы. С этой целью, так же,как и в предыдущем случае, вводится вспомогательная переменная *R,* в которую временно помещается одна из переставляемых строк (например, *A(ij)):*

ДЛЯ j=1К*m*

R=A(*ij*)

A(*ij*)=A(*i*+*kj*)

A(*i*+*kj*)=*R*

СЛЕДУЮЩИЙj

7. *Преобразование матрицы в одномерный массив.* Обработка одномерных массивов осуществляется быстрее, чем двумерных того же размера, что часто требует выполнения указанного преобразования.

Требуется переслать элементы матрицы размера*(n*x *m)* в одномерный массив того же размера по строкам с сохранением порядка следования элементов. Задача заключается в представлении исходной матрицы*A* в виде

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

вектора - столбца *X,* содержащего все элементы матрицы. Для этого нужно согласовать индексы исходной матрицыA и вектора Xпо формуле: $X((i\text{-}1)m+j)=A(ij)$.

В результате получим:

i=1, j=1: X(1) =A(1,1);

i=1, j=2: X(2)=A(1,2);

…

i=1, j=m: X(m)=A(1 ,m);

i=2,j=1; X(m+1)=A(2,1);

…

i=2,j=m; X(2m)=A(2,m);

…

i=n,j= 1; X((n-1)m+1)=A(n,1);

…

i=n,j=m; X(nxm)=A(n,m).

Этазадачарешаетсяс помощью следующих предписаний:

ДЛЯ i=1 К n

ДЛЯ j=1 *К* m

$X((i\text{-}1)m+j)=A(ij)$

СЛЕДУЮЩИЙ j

СЛЕДУЮЩИЙ i.

8. *Транспонирование матрицы.* Необходимо заменить строки матрицы ее столбцами, а столбцы - строками, т.е. вычислить $b_{ij}=a_{ij}$ где i=1,...,n;j=1,…, m:

ДЛЯ i=1 К n

ДЛЯ j=1 К m

$B(j,i)=A(ij)$

СЛЕДУЮЩИЙ j

СЛЕДУЮЩИЙ i.

Транспонированную матрицу можно получить в исходном массиве *A*. Для квадратной матрицы размером *(n*x*n)* нужно поменять местами каждый элемент верхнего треугольника с соответствующим элементом нижнего (диагональные элементы переставлять не следует). С этой целью используем прием перестановки строк матрицы (пункт 6) с помощью вспомогательной переменной *R*:

$R=a_{ij}$, $a_{ij}=a_{ij}$, $a_{ij}=R$,где$i=1,2,...,n-1$, $j=i+1,...,n$.

Эта задача решается с помощью следующих предписаний:

ДЛЯ *i*=1 К *n*-1

ДЛЯ *j*=*i*+1 К *n*

R=*A*(*ij)*

A(*i,j*)=*A*(*j,i*)

A(*j,i*)=*R*

СЛЕДУЮЩИЙ *j*

СЛЕДУЮЩИЙ *i.a*

Рассмотренные выше задачи относятся к одномерным и двумерным массивам. Однако в практике решения инженерных задач возможно обращение и к массивам большей размерности. В общем смысле многомерный массив в языке Паскаль трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов). Например, матрицу данных размером 10*12 можно хранить в массиве, описанном следующим образом:

Var H: Array [1..10] Of [1..12] Of Real.

Более привычной записью такого массива будет:

Var H: Array [1.. 10,1..12] Of Real.

Продолжая по аналогии, можно определить трехмерный массив как одномерный массив, у которого элементами являются двумерные массивы. Вот пример описания трехмерного массива:

Var T: Array [1..10,1..20,1..30] Of Real.

Этомассив, состоящий из 10*20*30, т. е. 6000 вещественных чисел.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Глава 4. Алгоритмы управления технологическими процессами

Технологические процессы многочисленны и разнообразны. К факторам, обуславливающим это разнообразие, относятся вид используемого сырья, форма и количество необходимых энергоресурсов, количество стадий преобразования сырья, временные характеристики стадий процесса, вид готовой продукции.

По своему характеру процесс может быть непрерывным периодическим или дискретным. Непрерывным является процесс, в котором конечный продукт вырабатывается, пока подводится сырье, энергия, химикаты и т.д. Например, непрерывными являются различные процессы в целлюлозно-бумажной промышленности (ЦБП): непрерывная варка

целлюлозы в аппаратах «Камюр», отбелка целлюлозы, производство бумаги на бумагоделательных машинах и др. В качестве другого примера можно привести процессы нефтепереработки и т.п.

В периодическом процессе в течение сравнительно короткого промежутка времени - часов или дней- вырабатывается определенное, ограниченное количество конечного продукта. Сырье и полуфабрикаты вводятся регламентированными дозами в заданной последовательности; операции смешения и подачи энергии осуществляются в заданном порядке. После того как операции выполнены в соответствии с заданной программой, получается порция конечного продукта. Хорошо известный пример такого процесса- это доменный процесс производства стали или в ЦБП это процесс периодической варки целлюлозы.

Дискретные процессы в первую очередь ассоциируются с изготовлением и обработкой деталей узлов и готовых изделий. Характерным примером дискретного процесса может служить процесс по сборке автомобилей или, в нашем случае, процессы транспортировки и обработки баланса на лесной

бирже, упаковки и транспортировки кип целлюлозы, изготовление тетрадей и многие другие процессы.

В заключение необходимо отметить, что алгоритмы управления, которые будут рассмотрены в следующих разделах, не привязаны однозначно к характеру процесса. В частности, при реализации отдельных задач управления непрерывными технологическими процессами, используются алгоритмы, характерные для управления дискретными процессами и. т. д.

Системы управления классифицируются по различным признакам, одним из них является форма представления информации в системе (формы сигналов), см. рис.4.1.

Алгоритмы аналогового и цифрового управления будут рассматриваться в разделе управления непрерывными технологическими процессами. Двоичные сигналы используются в алгоритмах управления дискретными процессами.

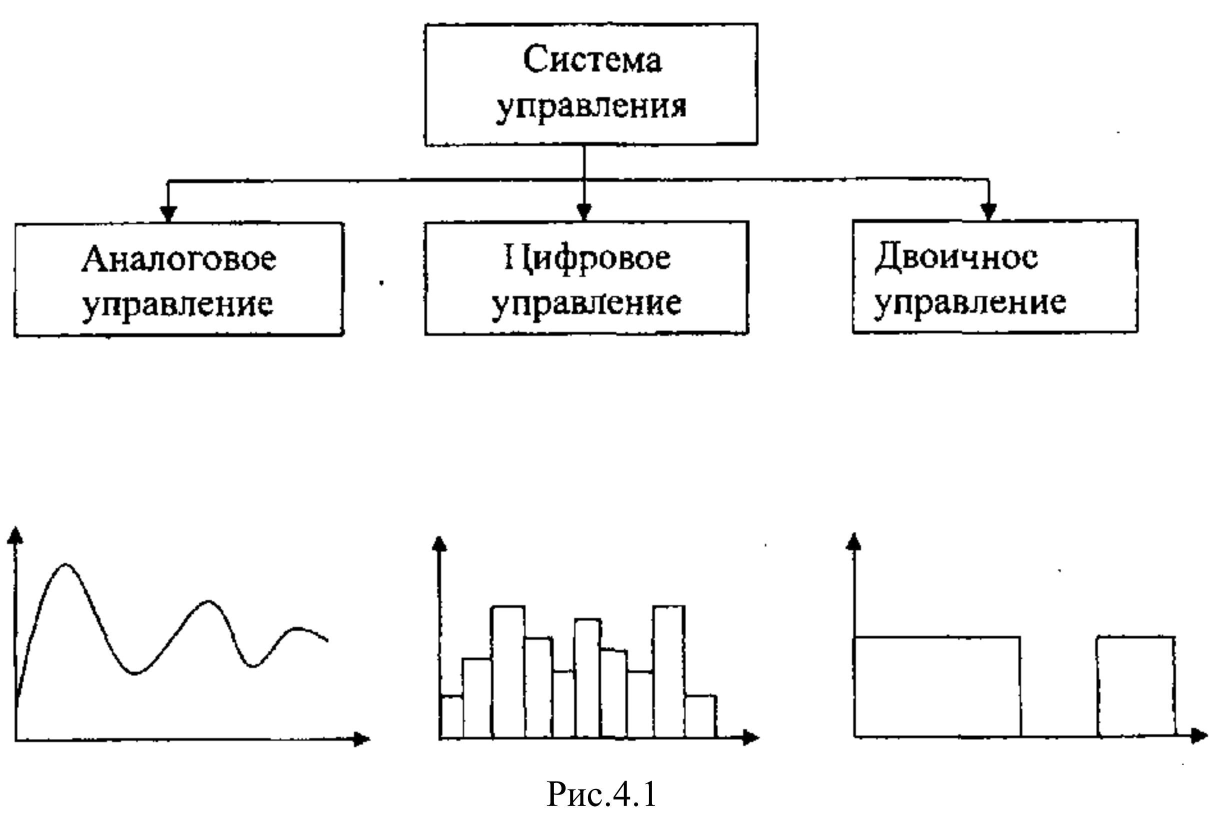


Рис.4.1

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

4.1. Алгоритмы управления дискретными процессами

Для описания алгоритмов управления дискретными процессами используются двоичные переменные (сигналы), которые принимают значенияО или 1 (один бит информации). Преобразование переменных осуществляется в соответствии с правилами булевой алгебры. Используя логические переменные, технологические операции могут быть представлены в форме логических высказываний. Высказывание является предположением, которое может быть или истинным или ложным. Истинное высказывание обозначается 1, ложное О.Покажем, как простейшие условия включения в работу массного насоса могут быть описаны с помощью логических высказываний. Принципиальная схема управления насосом дана на рис. 4.2 Оператор может включить насос по месту или дистанционно. Дня этого необходимо установить ключ выбора режима SA в положение “М” или “Д”. Если ключ занимает нулевое положение, насос не может быть включен. После нажатия одной из кнопок "ПУСК" (SB1 или SB2) включается электромагнит К1 масляного редуктора в линии смазки подшипников насоса. Для включения пускателя двигателя насоса должны быть выполнены два условия: уровень в массном бассейне выше минимального: L>Lo, и давление в линии смазки подшипников не ниже нормы: P>Po. На рис. 4.2 показаны сигнализаторы уровня и давления SLиSP.

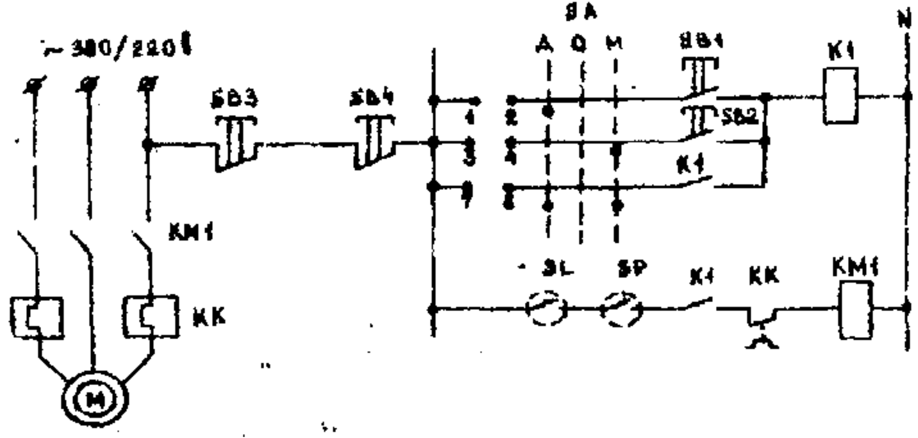


Рис 4.2

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Рассмотрим предложение, состоящее из двух простых высказываний: насос включается, если давление в масляном редукторе P>=Po (первое простое высказывание) и если уровень в массном бассейне L>Lo (второе простое высказывание). Обозначим первое высказывание X1, а второе X2.

Возможны следующие четыре случая:

1. P<Po (X1=0) , L<= Lo (X2=0)
2. P<Po (XI =0) , L>Lo (X2=l) (4.1)
3. P>Po (X1=1) , L<= Lo (X2-0)
4. P>Po (X1=1) , L>Lo (X2=l).

В первых трех случаях одно из простых высказываний ложно и поэтому ложно и более сложное предложение, составленное из этих высказываний. В последнем случае оба высказывания истинны, истинным является и сложное предложение (условия включения насоса выполнены). Таким образом, результат сложного высказывания является функцией двух простых. Если обозначить сложное высказывание Y, то эта функция записывается

Y= X1&X2 (4.2)

и представляет собой конъюнкцию (логическое умножение) - одну из элементарных логических функций.

Рассмотрим ещё одно предложение: насос может быть включен, если ключ выбора режима SA будет в положении "M" (первое простое высказывание- - X1) или в положении “Д” (второе простое высказывание - X2). Возможны следующие сочетания XI и X2:

1. SA в нулевом положении (X1=0, X2=0);
2. SA в положении "M" (X1=1, X2=0); (4.3)
3. SA в положении "Д" (X1=0, X2=1);
4. SA одновременно в положении "M" и "Д" быть не может, и это сочетание рассматривается лишь теоретически (X1=1, X2=1).

Очевидно, что сложное высказывание истинно во всех случаях, кроме первого (насос нельзя включить, если ключ в нулевом положении).

Обозначив сложное высказывание Y, запишем функцию

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

$$Y = X1 \vee X2, \quad (4.4)$$

называемую -дизъюнкцией (логическое сложение).

Рассмотрим ситуацию, описываемую одним простым высказыванием, например "двигатель включен" (X). Это пример логической функции

$$Y = X, \quad (4.5)$$

зависящей от одной логической переменной.

Ситуация “двигатель не включен “ образуется из предыдущего высказывания отрицанием “НЕ”:

$$Y = not(X). \quad (4.6)$$

Булевы операции используются для описания алгоритмов управления дискретными процессами наряду с релейно-контактными схемами, которые рассмотрены в последующих разделах пособия.

4.1.1. Алгоритмы управления по времени

Автоматическое управление в функции времени является одним из вариантов последовательного управления. Это.управление с принудительным пошаговым процессом, при котором переключение программы от шага к шагу зависит от определенных условий, выполняемых в ходе процесса. В качестве этих условий переключения используются определенные моменты времени. Алгоритм такого управления можно представить в виде двух частей: алгоритм формирования

последовательности интервалов времени и алгоритм управления исполнительными устройствами в соответствии с интервалами времени. Формирование последовательности интервалов времени выполняется с использованием устройств создания выдержек времени (реле времени или таймеров) или счетчика последовательности импульсов заданной длительности.

Рассмотрим формирование последовательности интервалов времени с помощью таймеров, которые составляют генератор интервалов времени (ГИВ). Программная реализация ГИВ на языке релейно-контактных схем приведена на рис.4.3. В рассматриваемой программе используются таймеры

с задержкой на включение. При описании алгоритмов управления будет использован язык релейно- контактных схем.

Рис.4.3

При установке бита «Запуск» в единичное состояние (ON) таймеры последовательно формируют интервалы времени, а при установке в нулевое состояние (OFF) все таймеры сбрасываются в течение одного цикла сканирования программы. Сам бит «Запуск» имеет нулевой интервал времени от начала запуска ГИВ.

Управляющие сигналы на исполнительные устройства формируются в функции состояния бита «Запуск» и флагов таймеров ГИВ. Например, в программе, приведенной на рис.4.4, бит ИУ (исполнительное устройство) включается при установке бита «Запуск» в ON и выключается после отработки таймера TIM2, а также включается после отработки таймера TIMi и выключается после отработки таймера TIMj (рис.4.4).

Рис. 4.4

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Рассмотрим классический пример, простую программу уличного светофора для управления дорожным движением. Исполнительными устройствами здесь являются лампы красного, желтого и зеленого огней светофора. Включается светофор управляющим сигналом «Работа».

Диаграмма управления светофором показана на рис.4.5 и осуществляется в соответствии со следующими периодически повторяющимися интервалами времени: T1 - зеленый свет, T2 - желтый свет, T3 - красный свет и T4 - красный и желтый свет.

Программа ГИВ светофора приведена на рис.4.6 и рис.4.7. Обращаем внимание, что после окончания интервала времени T4 флаг таймера TIM4 устанавливает бит запуска в OFF, при этом все таймеры сбрасываются (в том числе и таймер TIM4), после чего работа ГИВ повторяется.

Временная диаграмма работы ГИВ и исполнительных устройств светофора показана на рис.4.5. Бит «Работа» постоянно находится в состоянии ON.

Рис.4.5

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

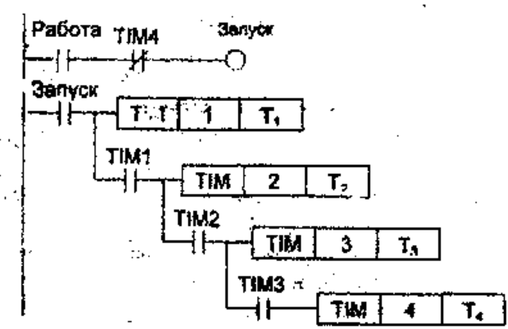


Рис.4.6

TIM2 Красный
TIM1 TIM2 Желтый
TIM3
Запуск TIM1 Зеленый

Рис.4.7

4.2.1. Алгоритмы жесткого последовательного управления по состоянию процесса

Автоматическое управление в функции состояния объекта при жесткой последовательности операций также является одним из распространенных способов управления в дискретной автоматике. Последовательность операций может выполняться однократно по одной команде или повторяться многократно в повторяющихся циклах. Алгоритм такого управления можно представить в виде двух частей:

– алгоритм формирования последовательности действий или шагов управления (отдельных операций, выполняемых в объекте управления);

– алгоритм управления исполнительными устройствами в соответствии с шагом управления.

Последовательность шагов (отдельных операций, выполняемых в объекте управления) формируется с использованием датчиков состояния объекта, которые информируют систему управления об окончании текущей

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

операции. Факт окончания предыдущей операции является необходимым условием начала следующей.

Различные шаги управления могут использовать одни и те же датчики или исполнительные устройства, поэтому необходимо фиксировать шаги. В этом случае работа датчиков на последующих шагах не будет влиять на предыдущие и, соответственно, на управляемые ими исполнительные механизмы. Для этого в программе каждый шаг управления связывается с битовой переменной (признаком шага), которая в момент активизации шага устанавливается в единичное состояние ON.

Таким образом, последовательность действий в объекте управления формируется программой, которая генерирует ряд шагов, последовательно устанавливая связанные с ними биты. По окончании последнего шага все признаки шагов сбрасываются. Если цикл необходимо повторить, то последний шаг должен опять запустить программу последовательности шагов.

Управление исполнительными механизмами определяется текущим шагом. Действие указывается для того механизма, который на этом шаге включается или выключается. Исходя из этого, формируется функция управления исполнительными устройствами: для каждого устройства определяется, на каком шаге оно включается, на каком выключается.

Рассмотрим пример управления позиционным манипулятором, схема которого представлена на рис.4.8. Манипулятор имеет гидравлический привод подъема и перемещения захвата и пневматический привод разжима захвата. Управление -золотниками гидроприводов осуществляется с помощью электромагнитов: VI – подъем,V2 - опускание. V3 - выдвижение, V4 - втягивание, V5 разжим захвата (зажим захвата выполняется пружиной). Контроль состояния приводов манипулятора выполняется конечными выключателями: SQ1 - поднят.SQ2 - опущен, SQ3 - выдвинут, SQ4 - втянут, SQ5 - захват разжат. Зажатое состояние захвата определяется контактным

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

датчиком давления PS по отсутствию давления в приводе захвата. Исходное состояние манипулятора - захват поднят над местом А.

Рассмотрим программу взятия изделия с места В и его установку на место А, которая выполняется однократно после подачи команды «Переставить В - А». Траектория движения захвата показана пунктирной линией на рис.4.8. Для однократного выполнения действий операции перестановки изделия с места В на место А по изменению состояния бита команды «Переставить В - А» из OFF в ON (по переднему фронту сигнала) формируется дифференциальный сигнал-бит «Диф, В - А» (рис.4.9), который устанавливается на время одного сканирования программы.

Если манипулятор не находится в исходном состоянии, то устанавливается бит сообщения Wm «Манипулятор не в исходном состоянии» и ставится на самоудержание через инверсный сигнал-бит RstW сброса аварийных и предупредительных сообщений (рис. 4.10). После приведения манипулятора в исходное положение (захват поднят и втянут) необходимо кратковременно подать сигнал RstW.

Рис.4.8

Если манипулятор в исходном состоянии, то начинает работать программа последовательности шагов, приведенная на рис.4.11. По сигналу-биту «Диф.

В - А» устанавливается флаг выполнения команды «Исп. В - А. Шаг1».который одновременно является первым шагом в программе. Установившись, флаг «Исп В - А. Шаг1» становится на самоудержание. Затем последовательно, по мере выполнения движений исполнительные механизмами и срабатывания датчиков устанавливаются остальные шаги программы. Каждый предыдущий шаг разрешает установку следующего. Установка бита «Шаг8» сбрасывает все шаги программы.

Формирование управляющих сигналов на исполнительные устройства приведено на рис.4.12

Рис. 4.11

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

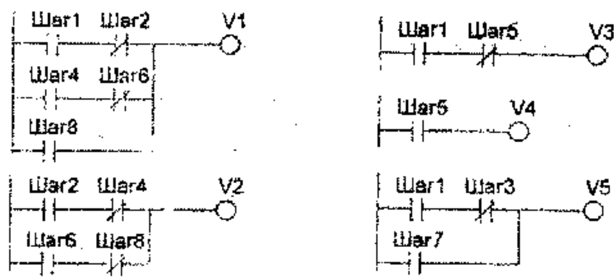


Рис.4.12

4.1.3. Алгоритмы гибкого управления по состоянию процесса

Для систем дискретного автоматического управления объектами конечным числом состояний, у которых алгоритм перехода из одного состояния в другое определяется значениями параметров объекта и не имеет жесткой последовательности, могут быть использованы известные алгоритмы цифровых автоматов, например автомата Мура.

Блок-схема автомата Мура представлена на рис. 4.13 , где Y - вектор выхода, S- вектор текущего состояния, S’ - вектор нового состояния, X- вектор входа, B-функция выхода, M - память, A- функция перехода. Если алгоритм управления конкретным объектом строить на основе алгоритма автомата Мура, то векторы будут иметь конкретное содержание. Вектор

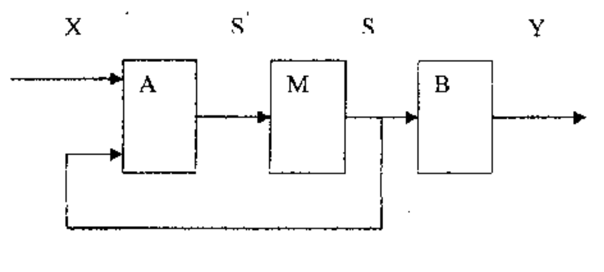


Рис.4.13

выхода Y определяет совокупность сигналов управления на исполнительные устройства объекта управления.

Векторы состояния S и S' отражают режимы работы объекта и его отдельных частей. Вектор входа X соответствует совокупности внешних сигналов управления, влиянию возмущающих воздействий на объект

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

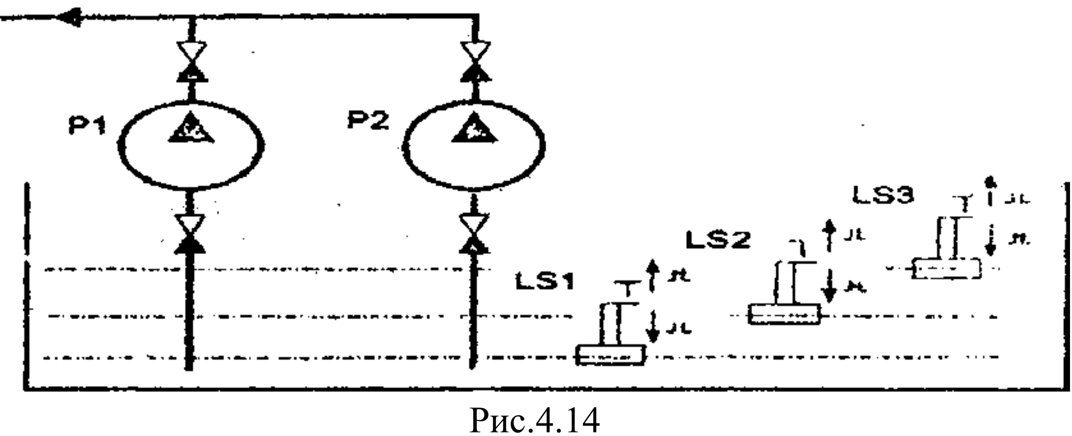


Рис.4.14

управления и состоянию текущих параметров самого объект управления.

Рассмотрит пример составления алгоритма и управляющей программы для системы управления насосной станцией осушительной системы, состоящей из двух насосных агрегатов P1 и P2 (рис.4.14). Насосы могут работать как по одному (при малых поступлениях воды), так и вместе (при больших поступлениях воды). Контроль поступления воды осуществляется тремя дискретными датчиками уровня LS1, LS2 и LS3. Обращаем внимание, что датчик уровня LS1 выдает прямой сигнал: вода выше контролируемого уровня - единица, ниже - нуль, а датчики LS2 и LS3 выдают инверсный сигнал. При низком уровне воды (ниже положения датчика LS1) насосы не работают. Для равномерной загрузки насосных агрегатов ведется учет времени работы каждого агрегата.

Насосная станция имеет следующий технологический процесс: - насос включается, если есть разрешение на его работу;

- разрешение на работу насоса устанавливается переключателем на пульте управления, снимается этим же переключателем или (при срабатывании защиты насоса) системой защиты;
- если уровень воды поднялся выше положения датчика LS2, то включается один насос, который имеет меньшее количество наработанных часов;

если уровень воды поднялся выше положения датчика LS3, то

включается второй насос, т. е. работают оба насоса;

- если уровень воды опустился ниже положения датчика LS2, то один из насосов выключается, остается в работе один насос с меньшим количеством наработанных часов;

- если уровень воды опустился ниже положения датчика LS1, то последний работающий насос выключается.

Насосная станция как объект управления имеет 4 состояния: S0 - оба насоса выключены, S1 - включен первый насос, S2 - включен второй насос, S3 - оба насоса включены. Таким образом, вектор состояния имеет вид S={S3.S2, S1, S0} и может быть представлен двоичным числом, где значения S3.S2. S1.S0представлены значениям соответствующих двоичных разрядов. Обязательным является условие, что один из этих разрядов всегда должен иметь единичное значение, причем только один из них.

Насосная станция имеет два исполнительных устройства: первый (P1) и второй (P2) насосы. Поэтому вектор управления для рассматриваемого объекта управления имеет вид Y={P2, P1} и его компоненты могут быть представлены битами. Значения этих бит определяются текущим состоянием объекта.

Функция выхода для системы управления насосной станцией может быть задана в табличной форме (табл.4.1), где задается зависимость выхода от состояния объекта.

Вектор входа включает сигналы разрешения работы первого и второго насосов H1 и H2, датчиков уровня воды в осушительной яме LS1,LS2 и LS3, признак сравнения времени, отработанного каждым насосом Fk. Признак Fk равен единице, если второй насос отработал больше времени, чем первый, и нулю, если время работы второго насоса не превышает времени первого. Таким образом, вектор входа имеет вид X={H1 H2, LS1,LS2, LS3, Fk}. Все элементы вектора входа могут быть представлены битовыми переменными. Примем, что разрешение на работу насосов и информация с датчиков о

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

превышении установленного для них уровня даются единичным значением сигнала.

Функция перехода определяет условия перехода системы из одного состояния (текущего, S) в другое (новое, S'). Наглядно функция перехода изображается графами (рис.4.15) или таблицами (табл.4.2).

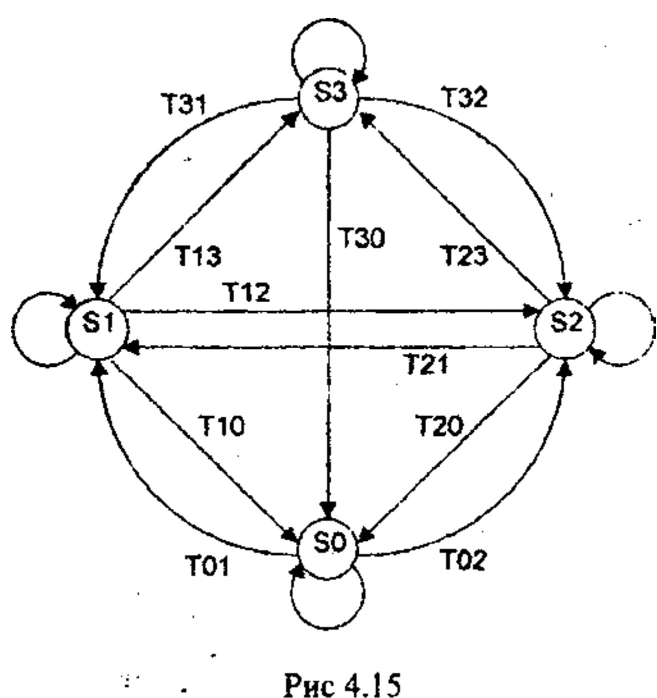


Рис 4.15

При описании функции перехода графом, состояния объекта представляются вершинами, а переходы в состояние дугами. Условие переходов записываются логическими выражениями

Таблица 4.1

S	Y	
	P1	P2
S0	0	0
S1	1	0
S2	0	1
S3	1	1

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Например, выключение обоих двигателей, т. е. переход в состояние S0 из состояния S3, S1 или 82, выполняется, если сняты разрешения на работу для обоих насосов или уровень воды опустился ниже LS1:

T30 = T10 = T20 =(notH 1) & (notH2)V(notLS 1).

Если разрешена работа двум насосам, но давление обеспечивается одним, то запрещение работающего насоса (например, при срабатывании защиты) приводит к запуску, другого, т. е. переходы из S1 в S2 и из S2 в S1 имеют следующий вид:

T12 = H1& (not H2); T21 = H1& (notH2).

Переход из состояния S3 в S2 выполняется, если снято разрешение насосу P1 или если оно есть, но уровень воды опустился ниже LS2 и насос P2 имеет меньшее наработанное время (Fk=0):

T32 = H2 & ((not H1) V (H1& LS2 & (not Fk))).

Аналогично записываются условия для остальных переходов.

При табличном представлении функции перехода (табл. 4.2) условие перехода записывается в графу на пересечении строки текущего состояния и столбца нового состояния. Когда объект находится в каком-либо состоянии, то в памяти записан соответствующий ему текущий вектор состояния. Если при этом условии ни одного из переходов, соответствующих этому состоянию, не выполняются, то вектор нового состояния равен нулю. Это означает, что перехода в новое состояние не требуется. Если выполнится условие какого-либо перехода, то соответствующий элемент вектора нового состояния будет установлен. Это означает, что требуется переход и его надо записать в память как текущее состояние. При этом изменится вектор управления и переведет объект в новое состояние. Далее будут проверяться условия переходов из этого состояния.

В реальных технических системах переход в новое состояние может потребовать небольшой паузы по времени или промежуточного контроля исходного состояния отключаемых исполнительных устройств. В ряде

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

случаев это бывает очень важно и необходимо предусмотреть дополнительные аппаратные и программные решения.

Управляющая пограмма на языке релейно-контактных схем приведена на рис.4.16-4.20 . На рис.4.16 показана начальная установка состояния объекта при включении системы управления и защита от неопределенности состояния. На рис.4.17 дана реализация функции выхода. На рис.4.18 приведена программа формирования нового-состояния с контролем и взаимным запрещением элементов (чтобы только один из них имел нулевое значение). Программа перехода системы в новое состояние (рис.4.19) выполняется, если есть требование перехода, т. е. один из элементов вектора S' не равен нулю, и заключается в присвоении нового значения вектору текущего состояния S. На рис.4.20 показана программа проверки условий и формирования переходов в соответствии с табл.4.2. Программы учета количества отработанных часов для насосных агрегатов, их сравнения и установки флага Fk здесь не приводятся.

Таблица 4.2

Текущее состояние	Новое состояние			
	S1	S2	S3	S4
S0		T01 H1&(notLS2)& ((notH2)V(H2&Fk))	T02 H2&(notLS2)& ((notH1)V(H1&(not(Fk)))	
S1	T10 (notH1)& (notH2)V (notLS1)		T12 H2&(notH1)	T13 H1&H2& (notLS3)
S2	T20 (notH1)& (notH2)V (notLS1)	T21 H1&(notH2)		T23 H1&H2& (notLS3)
S3	T30 (notH1)& (notH2)V (notLS1)	T31 H1&((notH2)V (H2&LS2&Fk))	T32 H2&((notH1)V (H1&LS2&(notFk)))	

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

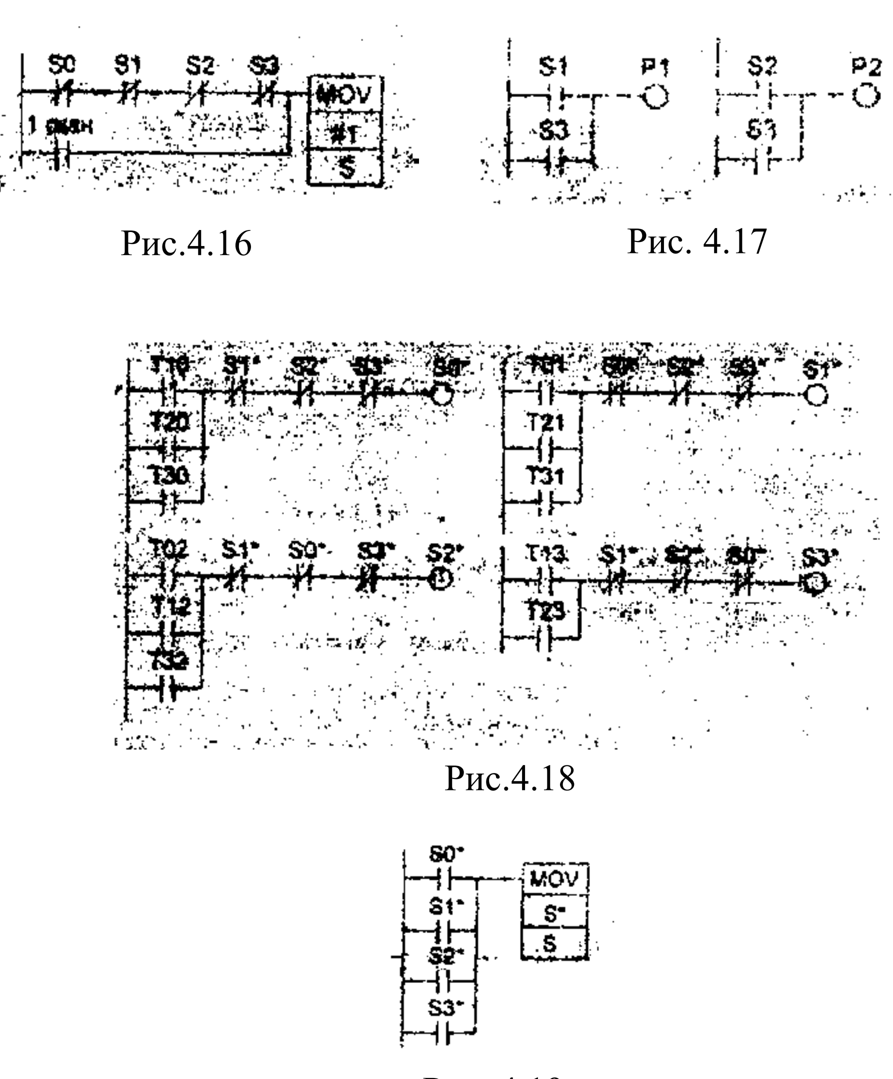


Рис.4.16

Рис. 4.17

Рис.4.18

Рис. 4.19

4.2. Алгоритмы управления непрерывными технологическими процессами

При рассмотрении алгоритмов управления не прерывными технологическими процессами необходимо выделить два аспекта: первый связан с технической реализацией системы управления, второй- с используемыми методами управления. В настоящее время большинство современных систем управления

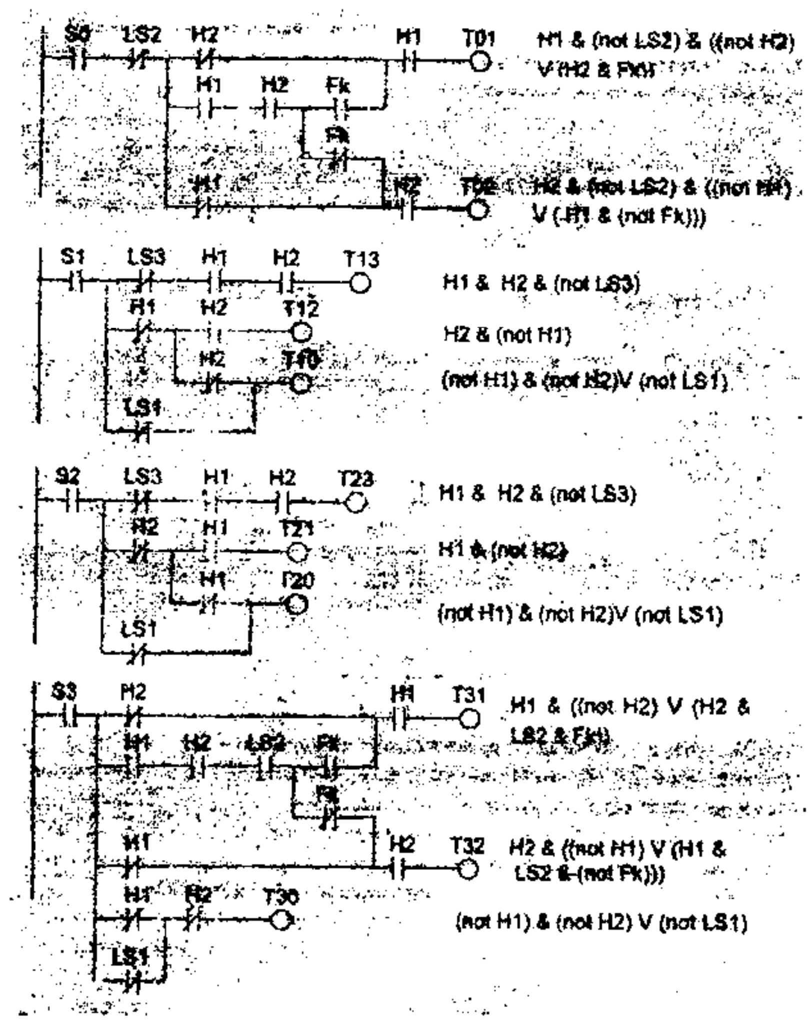


Рис. 4.20

реализуются в рамках автоматизированных систем управления на базе цифровых управляющих устройств. В тоже время информация, необходимая для управления непрерывными технологическими процессами, поступает с датчиков (давления, температуры и т.д.) в непрерывной форме. Для реализации цифровых алгоритмов управления информация должна быть преобразована в цифровую форму. Эти преобразования осуществляется рядом технических устройств и с помощью алгоритмов рассмотренных ниже. Наряду с использованием цифровых алгоритмов управления в разделе 4.2.4 рассматриваться и типовые законы управления в непрерывной форме.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Второй аспект связан с используемым методом управления. Теория

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

управления достаточно сложная наука и в настоящем пособии рассматриваются из алгоритмов управления только типовые алгоритмы управления, использующие принцип обратной связи.

4.2.1. Проверка достоверности данных и контроль отклонений

Проверка достоверности данных осуществляется для того чтобы устранить использование ошибочных данных, которые могут возникнуть в результате неправильного функционирования датчиков или других устройств в системах управления. Выделение ошибочных данных носит название проверки достоверности.Возможны также отклонения технологического режима, которые могут привести к нежелательным последствиям и которые также необходимо выявить. Контроль режима процесса носит название контроля отклонений. Проверка достоверности информации заключается в использовании верхних и нижних пределов, установленных для каждой технологической переменной. Например, при использовании преобразователей с унифицированным токовым сигналом 4-20 мА значение 4мА соответствует началу шкалы датчика измеряемой переменной, а 20 мА- верхнему пределу измеряемой шкалы. Если значение сигнала с преобразователя меньше или больше данных значений, то информация недостоверна и необходимо сформировать сообщение оператору процесса о том, что данный датчик неисправен.

Если $I[i]<4$ mA, или $I[i]>20$mA,

«Информация недостоверна». (4.7)

Дополнительно достоверность информации с датчика можно проверить по скорости изменения переменной.

Если $(I[i]-I[i-i])/T>V^{max}$,

«Информация недостоверна» (4.8)

где $I[i]$,$I[i-1]$-текущее и предыдущее значение измеряемой переменной, мА. Т-период дискретности опроса датчика в системе, с;

V^{max}-заданное максимально возможное значение скорости изменения переменной мА/с.

Использование алгоритма контроля достоверности по скорости изменения переменной особенно эффективно для устранения случайных всплесков сигналов. В этом случае при превышении скорости изменения сигнала текущее значение переменной игнорируется и выходной сигнал принимается равным предыдущему значению измеряемой переменной. Если ситуация повторяется подряд несколько раз то формируется признак недостоверности информации. В большинстве технических требований к программам управления технологическим процессом предусматривается, что программа должна проверять технологические данные и предупреждать операторов об обнаружении ненормального технологического режима; эту функцию часто называют контролем процесса. Хотя основная функция такого контроля достаточно проста, необходимо разработать программу проверки пределов таким образом, чтобы избежать повторной сигнализации от помехи при колебании переменной около верхнего или нижнего предела. Проблема колеблющихся сигналов решается введением зоны нечувствительности для сигнала об отклонении (Рис.4.21). Признак недопустимого отклонения включается в точке AH, но считается, что нормальный режим не восстанавливается до тех пор, пока процесс не вернется к точке NH. Разность между величиной сигнала в точках AH и NH называется шириной зоны нечувствительности. То же самое можно сказать и о точках AL и NL.

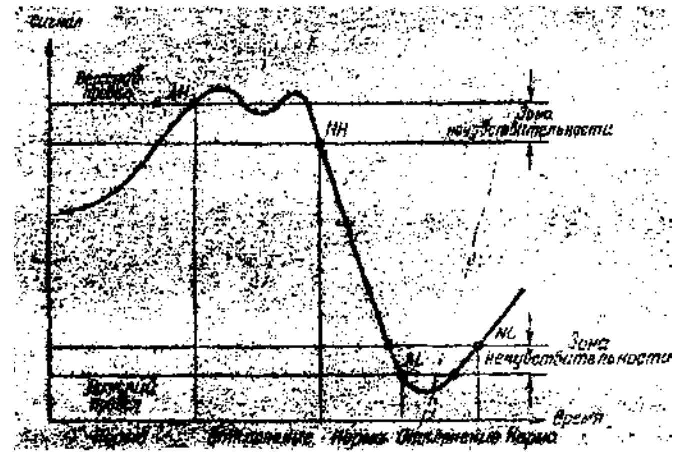


Рис.4.21

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Чтобы избежать сигнализации под воздействием помехи, обычно применяют одно- или двухразрядный двоичный счетчик и сигнал об отклонения возникает только после того, как два, три или четыре последовательных отчета оказываются за пределами нормы.

Такие повторные проверки особенно необходимы, когда сравнение с предельными значениями производится до цифровой фильтрации. Иногда полезно устанавливать две группы пределов. Внутренние пределы используются для предупреждения, внешние- как контрольные величины для сильного корректирующего воздействия.

Цифровые фильтры, описанные в следующем разделе, уменьшают колебания сигнала переменной, но одновременно снижают скорость реакции на большие изменения сигнала в результате выхода датчиков из строя. Поэтому проверка достоверности, как правило, должна производиться до цифровой фильтрации, а контроль пределов - после.

4.2.2. Алгоритмы фильтрации

После проверки достоверности информации используются цифровые фильтры для устранения влияния помех. Чаще всего применяются два типа фильтров: усредняющий и экспоненциального сглаживания. На рис.4.22-4.23 и проиллюстрировано два вида усредняющих фильтров. На рисунках **T-** время между вычислениями выхода фильтра. В фильтре, показанном на рис.4.22 , используется равномерный опрос и требуется память для храненияп последних значений входа, т.е. выход фильтра является текущим средним значение, рассчитанным на основании **n**предыдущих измерений :

В фильтре, представленном на рис. 4.22 накапливаются п последовательных измерений перед вычислением выхода, и новое значение Y[i] с выхода фильтра выдается тогда, когда будут получены все **n** отсчетов.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

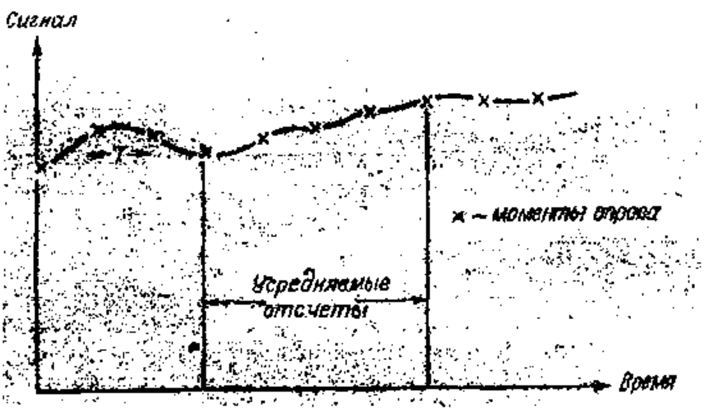


Рис. 4.22

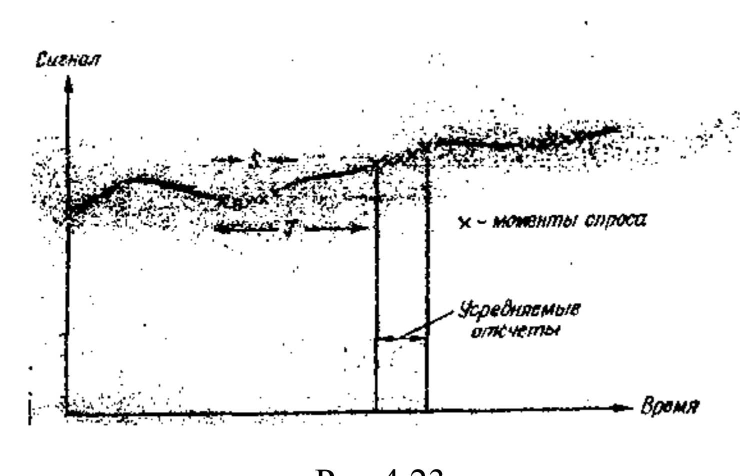


Рис.4.23

Во втором фильтре для накопления измерений требуется лишь одно слово впамяти:

Главный недостаток неравномерного опроса заключается в неэффективном использовании аналого-цифрового преобразователя (АЦП).

Экспоненциальный фильтр программируется просто и дает меньшее искажение полезного сигнала, чем усредняющий фильтр. Это дискретный эквивалент инерционного звена. Выходной сигнал экспоненциального фильтра рассчитывается по формуле:

$$Y[i]=Y[i-1]+k^* \{X[i]-Y[i-1]\} \quad (4.11)$$

4.2.3. Алгоритмы масштабирования

После того как устранены помехи, необходимо преобразовать информацию в осмысленную форму. Цифровая информация после аналого-цифрового преобразователя является линейным отображением напряжения, приложенного к зажимам блоков аналогового ввода, чтобы представить информацию оперативному персоналу, управляющему процессом, в физических единицах (давление, температура, расход и т.д.) необходимо произвести некоторое преобразование. По крайней мере, почти всегда необходимо линейное преобразование шкалы для учета пределов измерения датчика. Обычно встречаются четыре типа преобразований: линейное, извлечение квадратного корня, преобразование сигналов температуры, снимаемой непосредственно с термопар, и преобразование сигналов температуры с выходов преобразователей.

Уровни и давления обычно измеряются линейными датчиками. Таким образом надо произвести только линейное преобразование шкалы из цифрового отсчета в отсчеты в физических единицах. Необходимые, для такого преобразования константы могут указаны либо путем задания значений измеряемой величины в технических единицах на концах шкалы, либо путем задания значения нуля центрального отсчета и самой шкалы. При этом используется следующие алгоритмы преобразования:

$$\mathbf{Y = YMIN + (X - XMIN)\frac{YMAX - YMIN}{XMAX - XMIN}} \qquad (4.12)$$

В первом выражении Y- значение величины в технических единицах, соответствующее величине X данного цифрового отсчета; YMAX и YMIN- значения в технических единицах на концах шкалы; XMIN и XMAX -

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

значения цифрового отсчета, соответствующие YMIN и YMAX.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Во втором выражении Y0-значение в технических единицах, соответствующие нулю цифрового отсчета.

Расходы обычно измеряются дифференциальными трансформаторами, в которых перепад давления пропорционален квадрату расхода

$$P = \sqrt{PMIN^2 + (X - XMIN)\frac{PMAX^2 - PMIN^2}{NMAX - NMIN}} \quad (4.14)$$

где P- текущее значение измеряемой величины;

PMIN, PMAX-минимальное и максимальное значения шкалы датчика, соответственно;

NMIN, NMAX-минимальное и максимальное значения цифрового отсчета; X-значение цифрового отсчета соответствующее измеряемой величине.

Данная формула является сложной для вычислений и иногда для преобразования используется итеративный алгоритм Ньютона-Рафсона. Число итераций можно уменьшить, если использовать значение, полученное в предыдущем опросе, в качестве первого приближения в последующем расчете.

При преобразовании сигналов температуры, снимаемой непосредственно с термопар, функция преобразования может быть задана в табличной форме. Обычно в практике управления технологическими процессами используют кусочно-линейную аппроксимацию этой таблицы. Количество используемых линейных отрезков зависит от требуемой точности и пределов измеряемых температур. Четыре или пяти линейных отрезков обычно достаточно для тех пределов изменения температур и точности, которые требуются для большинства автоматизированных систем управления.Иногда для преобразования используют полиномы более высоких порядков, например:

$$T=A_3X^3+A_2X^2+A_1X+A_0, \quad (4.15)$$

где T- текущее значения измеряемой переменной;

X- текущее значение цифрового отсчета.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

4.2.4. Алгоритмы типовых законов управления

Рассмотрим два основных метода управления. Управление по принципу обратной связи означает (упрощенно), что измерение производится после управляющего органа (см. рис. 4.24) и далее, в зависимости от результатов измерений, осуществляется регулирование.

Рис.4.24

Управление по возмущению (по прямой связи), (рис. 4.25) иногда называют управлением с упреждением. Управляющее воздействие прилагается «впереди» точки измерения. Данный метод позволяет вносить коррекцию в «будущей» (относительно измерения) точке процесса. Обратная связь изменяет режим в начале процесса до измерения в точке расположения датчика. При управлении по возмущению делается попытка заранее предвидеть изменения и настроить поток в точке, которая еще может оказать влияние на значение измеряемого параметра потока. Управление по возмущению особенно ценно в процессах со значительным временем отклика

(большая инерционность процесса, большое запаздывание). Управление по

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

обратной связи часто не в состоянии обеспечить надлежащее качество регулирования для таких процессов.

Рис.4.25

Наиболее часто в качестве алгоритма управления используются типовые законы управления. Эти законы управления могут быть описаны в непрерывном виде следующим уравнением:

$$U(t) = K_0 + K_k E(t) + \frac{K_R}{T_I}\int_0^t E(t)dt + K_R T_D \frac{dE(t)}{dt} , \qquad (4.16)$$

где K_0, K_K,T_I,T_D- параметры настройки, благодаря которым алгоритм управления (регулятор) можно настроить на работу во многих различных процессах;

E(t)- ошибка регулирования (в системах управления по возмущению вместо E(t) необходимо подставить F(t)), это разность:

$$E(t)=X(t)-Y(t), \qquad (4.17)$$

где X(t) -заданное значение;

У(t)-измеряемая переменная.

Уравнение регулятора может быть представлено в операторной форме:

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

$$X(t) \equiv X(p), \qquad \frac{dX(t)}{d(t)} \equiv pX(p), \qquad \int_0^1 X(t)dt \equiv \frac{X(p)}{p} \qquad (4.18)$$

В данном случае, Х-обобщенное обозначение переменной. Блок -схема (см. рис.4.26).

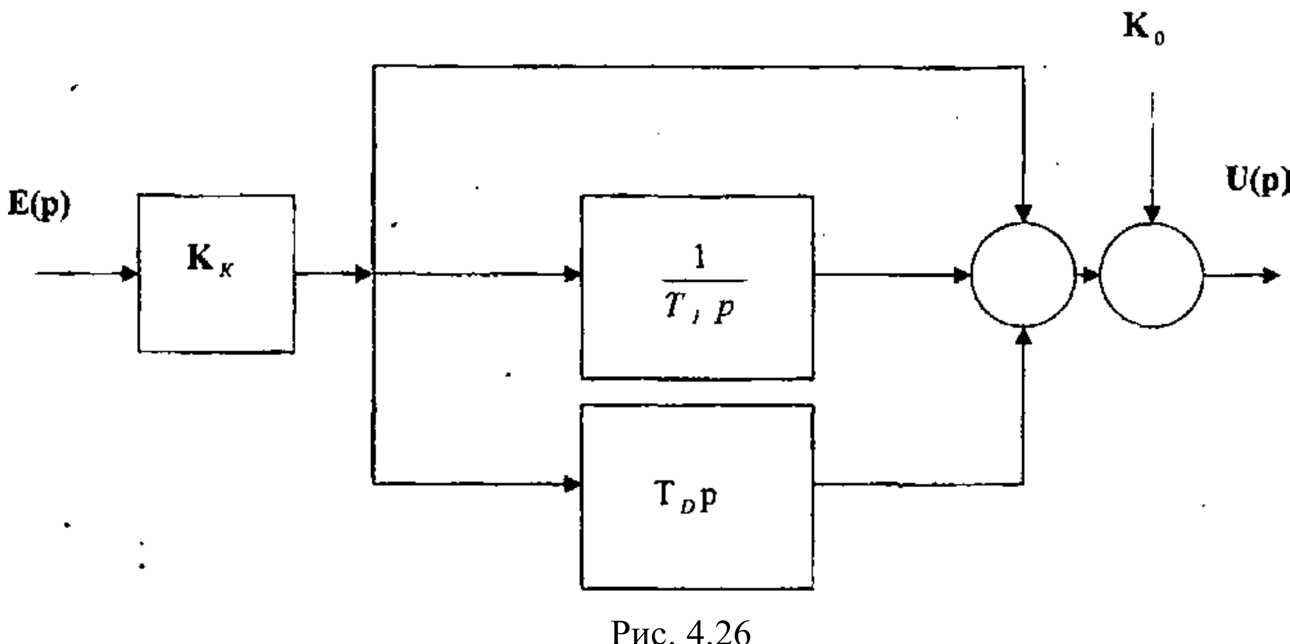


Рис. 4.26

Не каждый регулятор рассчитывает все слагаемые уравнения. Простейшие регуляторы, известные под названием «пропорциональных (сокращенно П)», используют только два первых слагаемых; если добавить третью составляющую, то регулятор называют « пропорционально интегральным (ПИ)»; регуляторы, использующие все члены, носят название пропорционально -интегрально -дифференциальных(ПИД)».Путем изменения параметров регулятор может быть настроен для решения самых разнообразных задач.

Для реализации алгоритма управления на базе цифровых управляющих машин необходимо от интегрально-дифференциального уравнения перейти к разностному уравнению. При «небольшом» значении периода опроса датчиков Т уравнение (4.16) можно преобразовать в разностное с помощью

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

замены производной разностью первого порядка и замене интеграла суммой по методу прямоугольников . В итоге преобразований мы получим разносное уравнение регулятора следующего вида:

$$U[(i)T]=U[(i-1)T]+q_0E[(i)T]+q_1E[(i-1)T]+q_2E[(i-2)T] \, , \qquad (4.19)$$

где

$$q_0 = K_K\left(1+\frac{T_D}{T}\right);$$

$$q_1 = -K_R\left(1+2\frac{T_D}{T}-\frac{T}{T_1}\right);$$

$$q_2 = K_K\frac{T_D}{T}$$

Другой вариант ПИД-регулятора представлен ниже (рис.4.27) на одном из пяти языков программирования контроллеров, а именно на графическом языке FBD (FunctionBlockDiagram). Схематично функциональный блок можно представить следующим образом рис.4.27.

SET_POINT-заданное значение,

ACTUAL-измеряемая переменная,

Y_OFFSET-значение переменной управления соответствующей стационарному режиму,

RESET-признак установки начального состояния регулятора,

Y-переменная управления,

TM- шаг квантования (период опроса датчиков),

TN- время интегрирования,

TV-время дифференцирования.

ПИД- регулятор в данном варианте реализации состоит из ряда других функциональных блоков. В блоке SUB производится вычитание измеренной величины из величины задания. Блок INTEGRAL реализует операцию интегрирования ошибки регулирования. В блоке DIV производится деление интеграла на время интегрирования TN. Блок DERIVATIVE реализует операцию вычисления производной от ошибки регулирования, далее значение производной умножается с помощью блока MUL на время

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

дифференцирования TV.

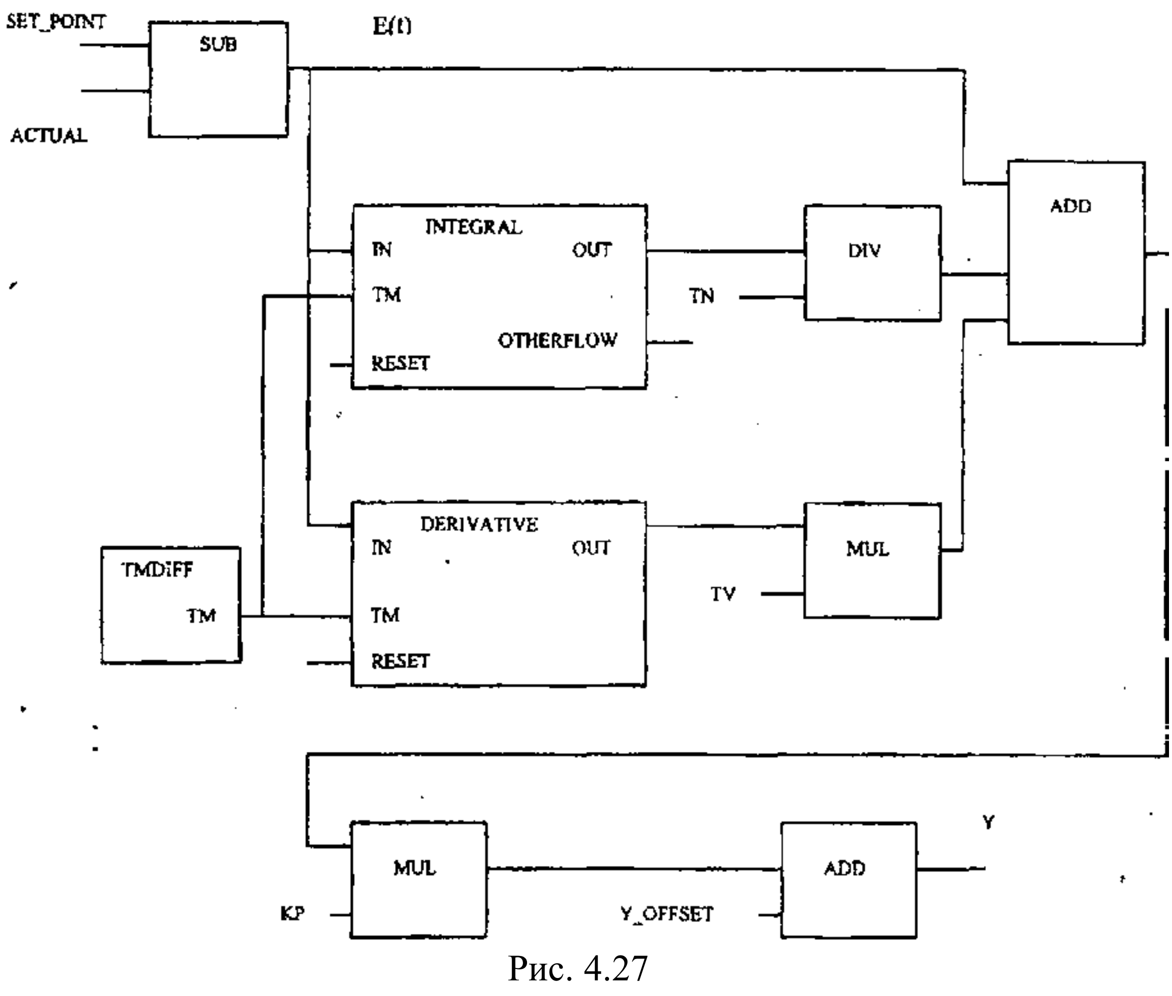


Рис. 4.27

Ошибка регулирования, интеграл и производная от ошибки регулирования, умноженные на соответствующие коэффициенты настройки регулятора, складываются с помощью блока ADD. Полученный суммарный сигнал умножается на коэффициент усиления регулятора KP с помощью блока MUL. Как правило, все системы регулирования работают около некоторого стационарного состояния, поэтому к выходной величине добавляется с помощью блока ADD значение переменной управления, соответствующей стационарному режиму.

Глава 5. Организация структур данных и файлов

5.1. Организация структур данных

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Одной из важных предпосылок успешного проектирования систем является четкое представление характера данных, которыми оперирует прикладная программа. Создатель языка Паскаль Н. Вирт в своей книге «Алгоритмы + структуры данных = программы» писал: «все интуитивно чувствуют, что данные предшествуют алгоритмам: необходимо иметь некоторые объекты, прежде чем оперировать ими».

Под структурами данных понимают наборы некоторым образом организованных данных. Характер организации может быть простым, например, когда элемент в массиве определяется своим номером (индексом) или сложным, например, когда элемент двухсвязного списка вместе с собственно данными должен содержать указатели преемника и предшественника. Программисту необходимо знать наиболее распространенные структуры данных, способы хранения их в памяти системы и эффективного использования в прикладных программах. Правильный выбор структуры данных позволяет уменьшить объем памяти и увеличить производительность системы. Расширение возможностей компьютеров позволяет использовать все более сложные структуры данных. Ниже кратко рассматриваются некоторые основные структуры данных.

Булевы переменные. Булевы (логические) переменные - это однобитовые переменные, принимающие одно из двух значений: 1 или 0 («Истина» или «Ложь»). В контроллерах систем управления достаточно много событий представляется булевыми переменными, кодирующими двоичные события в системе: «Вкл. -Выкл.»; «Открыто - Закрыто» и т.п. В контроллерах имеется набор операций над булевыми переменными для эффективной обработки логических переменных. К таким основным операциям относятся «Конъюнкция», «Дизъюнкция», «Исключающее ИЛИ», «Инверсия».

Целые числа. Представление целых чисел наиболее стандартизировано, обычно это байт, два байта, четыре байта. При ограниченной разрядности памяти контроллера данное может храниться в нескольких последовательно

расположенных ячейках. Целые числа, как типы данных, используются во многих языках программирования для описания алгоритмов.

Числа с плавающей запятой. Такое представление данных позволяет расширить диапазон по сравнению с целыми числами, а во-вторых, числа с плавающей запятой удобны для представления чисел с дробной частью.

Массивы. Наиболее простой и распространенной структурой данных является одномерный массив (рис.5.1).

Память

ADR	Элемент 0.
	Элемент 1
	Элемент 2
	.
	.
	Элемент N-1
	.

Рис. 5.1

Он представляет собой набор элементов данных одинаковой длины, который размещается в области смежных ячеек памяти с некоторого начального (базового) адреса ADR. Количество элементов в массиве называется его длиной, положение любого элемента в массиве характеризуется его порядковым номером, называемым индексом. Адрес элемента равен сумме базового адреса и индекса, умноженного на длину элемента в байтах. На практике целесообразно применять массивы, длина элементов которых кратна двум. Тогда при вычислении адреса элемента операция умножения заменяется сдвигами. В наиболее простых массивах длина элементов составляет 1 байт.

Формирование и обработка массивов осуществляются циклическими про-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

граммами, состоящими из трех частей:

- инициализации;
- обращения к текущему элементу;
- перехода к следующему элементу и проверки условия окончания цикла.

Более общую структуру данных представляет **двумерный массив**. Каждый элемент двумерного массива характеризуется двумя индексами: номером строки I и номером столбца J. Однако в памяти двумерный массив хранится в линейно упорядоченных смежных ячейках, что несколько усложняет обращение к текущему элементу. В массиве из однобайтных элементов общий индекс ind, который суммируется с базовым адресом, зависит от способа размещения массива:

$ind=i*n_r+ j$, если массив хранится по строкам;

$ind=i +j * n_c$, если массив хранится по столбцам, где n_r и n_c - число строк и столбцов в массиве. Когда длина элементов массива превышает 1 байт, значение ind следует умножить на длину.

Обработка двумерных массивов осуществляется программами, содержащими два цикла: внешний и внутренний. Во внешнем цикле производится инкремент (увеличение) переменной i, а во внутреннем (вложенном) - инкремент переменной j. При необходимости можно организовать массивы большей размерности. Вычисление общего индекса ind несколько усложняется, так как в памяти многомерные массивы по-прежнему хранятся линейно. Программы обработки многомерных массивов имеют соответствующее число вложенных циклов.

Очереди. Очередь представляет собой структуру данных, в которой элементы можно исключать только с одного конца (начало очереди), а включать только с другого (конец очереди). Главная особенность очереди заключается в том, что она сохраняет порядок элементов неизменным и такую структуру обычно называют FIFO (первый «приходит» - первый «уходит»).

Число элементов в очереди называется ее длиной. Обычно очередь иллюст-

рируют группой людей, ожидающих обслуживания. В машинной же реализации очереди элементы удобно сохранять "неподвижными", а ввести два указателя: начала и конца очереди. Первый адресует элемент, подлежащий исключению из очереди, а второй - последний элемент в очереди или, что иногда более удобно, ячейку сразу за последним элементом.

Наиболее часто в компьютерных системах очереди используются при вводе и выводе символьных данных. Для очереди выделяется некоторая область смежных ячеек памяти, число которых определяется максимально возможной длиной очереди. При организации очереди необходимо учитывать два особых случая. Попытка включить элемент в очередь, все ячейки которой уже заняты, называется переполнением, а попытка исключения элемента из пустой очереди называется антипереполнением. Обнаружение особых случаев значительно упрощается, если область очереди размещается на одной странице памяти, то есть, например, старшие 8 бит адресов всех ячеек очереди постоянны и сравнивать адреса можно по младшим 8 битам.

Для исключения элемента из очереди необходимо считать элемент, адресуемый указателем начала очереди, а затем произвести инкремент этого указателя (предполагается, что очередь «растет вниз», в область больших адресов). Включаемый в очередь элемент записывается в ячейку, адресуемую указателем конца очереди (удобнее, если этот указатель адресует первую свободную ячейку), после чего производится инкремент данного указателя. Разумеется, неограниченно увеличивать оба указателя нельзя. На рис. 5.2 приведена очередь из списка элементов P,Q,R,S,T. Два указателя обозначают ячейки текущих элементов очереди, первого и последнего. Для добавления нового элемента в очередь указатель « первый элемент» увеличивают на 1 и новый элемент помещают в массив «Имя». Чтобы удалить элемент из очереди, указатель «последний элемент» изменяют на 1 и тем самым элемент фактически удаляют из очереди. Такой подход с точки зрения доступа к элементам основаны на принцип е «первый вошел — первый вышел».

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

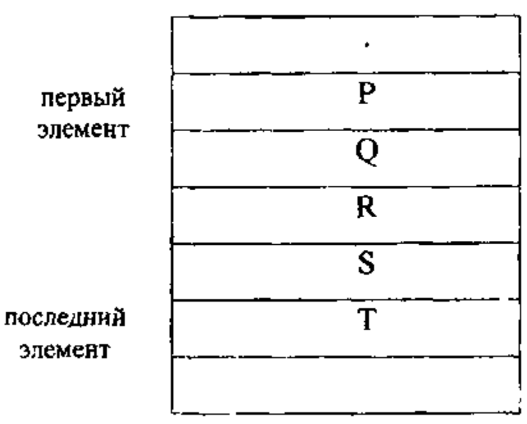


Рис. 5.2

Поскольку массив имеет конечную длину, указатели «первый элемент» и «последний элемент» рано или поздно принимают значения, соответствующие концу очереди. Элементы, расположенные в виде списка, сами могут быть сложными структурами. Работая, например, со списком массивов, мы на самом деле не добавляем и не удаляем массивы, ибо каждое добавление или удаление потребовало бы времени, пропорционального размерности массива. Вместо этого изменяем указатели массивов. Таким образом, сложную структуру можно добавить или удалить за фиксированное время, не зависящее от ее размера.

Списки. С математической точки зрения список - это конечная последовательность элементов некоторого множества. Описание алгоритма часто будет включать в себя некоторый список, к которому добавляются или из которого удаляются элементы. По этой причине необходимо создать структуру данных, которая позволит пользователю удалять или добавлять элементы в произвольном порядке, а не только из начала или конца массива элементов. Пусть задан список элементов: элемент 1, элемент 2, элемент 3, элемент 4. Простейшей его реализацией будет структура последовательно связанных элементов, представленная нарис. 5.3:

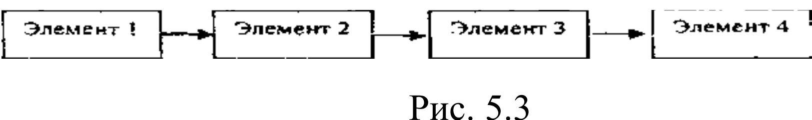


Рис. 5.3

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

В общем случае удаление или добавление новых элементов может быть эффективно выполнено из любого места списка, например из середины.

Каждый компонент в представленной структуре состоит из двух ячеек памяти. Первая ячейка содержит сам элемент, а вторая ячейка используется для хранения указателя следующего элемента. Это можно реализовать в виде двух массивов, которые на рис. 5.4 названы «Имя» и «Следующий»:

	Имя	Следующий
0	-	1
1	Элемент 1	3
2	Элемент 4	0
3	Элемент 2	4
4	Элемент 3	2

Рис. 5.4

В рассматриваемом массиве «Имя» - это хранящийся элемент, а «Следующий» - индекс следующего элемента в списке, если такой элемент существует. На рис. 5.4 «Следующий»[0] означает постоянный указатель на первую компоненту в списке. Заметим, что порядок элементов в массиве «Имя» не совпадает с их порядком в списке. Таким образом, структура данных типа список позволяет гибко удалять или добавлять новые элементы только за счет изменений адресов связанных элементов в структуре.

Стек. Стек представляет собой, как и очередь, специальную разновидность одномерного массива. Осуществлять загрузку элементов данных в стек и извлечение их из стека можно только с одного конца, называемого вершиной стека. Таким образом, в любой момент времени из стека можно считать только элемент, находящейся в его вершине и представляющий собой последнее загруженное в стек данное. Ячейку памяти, выполняющую роль вершины стека, адресует указатель стека SP(StackPointer). В процессорах, как правило, имеется аппаратный указатель стека SP и все команды, связанные

загрузкой данных в стек и извлечением их из стека, сопровождаются автоматической модификацией SP. До использования стека необходимо инициализировать SP для определения адреса вершины стека. В операциях со стеком могут возникнуть особые случаи, например попытка загрузить данные в ячейку, находящуюся вне области стека. Данная ситуация называется переполнением, а попытка извлечения данных из ячейки, адрес которой больше максимального адреса области стека, называется антипереполнением. В некоторых процессорах предусмотрены специальные средства контроля границ стека, в противном случае ответственность за это возлагается на программиста. Следует помнить, что команды вызовов подпрограмм используют стек автоматически. На рис.5.5 представлен стек, в который записано три элемента: элемент 1, элемент 2, элемент 3.

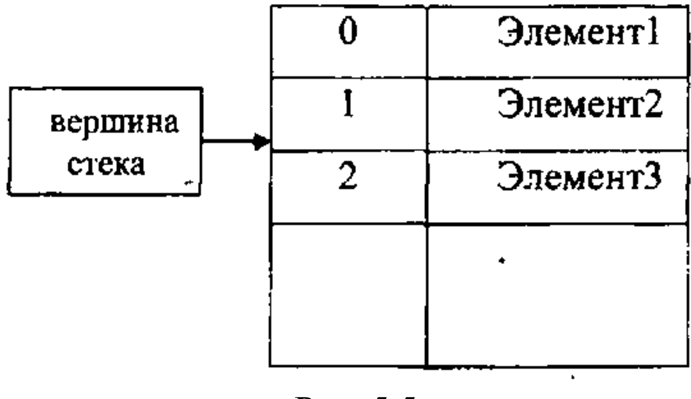


Рис. 5.5

Переменная «вершина стека» является указателем последнего элемента, записанного в стековую память. Чтобы добавить новый элемент в стек, значение указателя «вершина стека» увеличивается вначале на 1, а затем новый элемент записывается в следующую ячейку памяти массива. Так как емкость стека ограничена перед записью очередного элемента необходимо проверить значение указателя «вершина стека». Для удаления элемента из стека необходимо уменьшить значение указателя «вершина стека» на 1. Заметим, что физически удаляемый элемент стирать из стека не требуется, он стирается логически. Для определения количества элементов в стеке достаточно проверить значение указателя «вершина стека», которое должно соот-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ветствовать начальному значению указателя стека. Время загрузки нового элемента и извлечения элемента не зависят от числа элементов в стеке.

К более сложным структурам данных относятся двухсвязанные списки, деревья и другие структуры. Характерное отличие их от рассмотренных выше структур заключается в том, что каждый элемент наряду с собственно данными содержит минимум один указатель, например адрес следующего элемента (преемника). Необходимость дополнительной памяти для хранения указателей оправдывается возможностью хранения частей структуры в различных областях памяти и упрощением некоторых операций. Это, в свою очередь, обеспечивает динамическое распределение памяти системы. В прикладных программах контроллерных систем списки и деревья используются редко.

5.2. Организация файловой структуры

Файл - это именованная последовательность байтов команд или данных произвольной длины. Несмотря на то, что информация о местоположении файлов хранится в табличной структуре, пользователю она представляются в виде иерархической структуры, так как это для пользователей наглядней, а все необходимые преобразования берет на себя операционная система. К основным функциям файловой системы относятся следующие:

- создание файлов, каталогов (папок), присвоение им имен и их переименование;
- копирование и перемещение файлов между дисками компьютера и между каталогами (папками) одного диска;
- удаление файлов и каталогов (папок);
- навигация по файловой структуре с целью доступа к заданному файлу, каталогу (папке);
- управление атрибутами файлов.

Каталоги (папки) - важные элементы иерархической структуры, необходимые для обеспечения удобного доступа к файлам, если файлов на носителе слишком много. Файлы объединяются в каталоги по любому общему

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

признаку, заданному их создателем (по типу, по принадлежности, по назначению, по времени создания и т. п.). Каталоги низких уровней вкладываются в каталоги более высоких уровней и являются для них вложенными. Верхним уровнем вложенности иерархической структуры является **корневой каталог диска.**

В иерархических структурах данных адрес объекта задается обычно маршрутом (путем доступа), ведущим от вершины структуры к объекту. При записи пути доступа к файлу, проходящего через систему вложенных каталогов, все промежуточные каталоги разделяются между собой определенным символом. Во многих операционных системах в качестве такого символа используется «\» (обратная косая черта), например:

C:\Мои_документы \Текущие \Рефераты \Операционные_системы.doc.

Доступ к файлам. Наиболее распространенными являются следующие способы доступа к файлам:

- последовательный доступ (последовательные файлы);
- индексно-последовательный;
- прямой доступ (равнодоступные файлы).

Последовательная организация доступа предполагает размещение записей в файле последовательно одна за другой в порядке их поступления. Считывание записей возможно только в порядке их физического расположения. Этот метод является единственно возможным для организации файлов в томах с последовательной выборкой (магнитные ленты), но может использоваться и в томах с произвольной выборкой.

Для индексно-последовательного способа записи располагаются в логической последовательности в соответствии со значением ключей, являющихся элементами данных. Доступ к индексно-последовательным записям осуществляется последовательно, в порядке возрастания (убывания)значений ключа.

При прямой организации доступа к файлам каждой записи ставится в соответствие адрес ее размещения. Правила вычисления адреса и процеду-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ра ее определения задаются программистом. В файлах с такой организацией возможен прямой доступ к записи независимо от других. Метод прямой организации возможен только для устройств с прямым доступом к памяти (диски).

Организация работы с файлами. Файловая система обеспечивает хранение, выборку и совместное использование данных. Информация о размещение файлов находится в таблице, называемой таблицей размещения файлов FAT(FileAllocationTable). Каждый элемент FAT содержит имя файла, р'азмер записи, количество записей в файле, адрес первого блока, дату и время создания, атрибуты, кроме того, в FAT включается информация о свободной внешней памяти. Таблица размещения файлов,как правило, находится в том же томе,где и данные. Процедура обращения к файлу в общем случае состоит из следующих операций:

- создание файла или открытие существующего файла;
- запись в файл или чтение из файла;
- закрытие файла.

Защита данных. Обращение пользователя к файлу считается санкционированным, если обращение к этому файлу разрешено данному пользователю и задаваемая операция соответствует его полномочиям. Запись, содержащая ограничения на доступ к файлу и на полномочия (чтение и запись, только чтение, запуск программы, удаление файла и т.д.) называется ордером защиты. Для определения доступности файла могут применяться различные методы:

- задание паролей;
- задание таблицы управления доступом;
- шифрование.

Метод паролей. В структуру файла записывается некоторый цифровой или буквенно-цифровой пароль. При обращение к файлу пользователь должен указать пароль. Файл считается доступным, если указанный пароль совпадает с паролем ордера защиты.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Задание таблицы управления доступом.Права доступа задаются для каждого пользователя по отношению ко всем файлам с помощью таблицы доступа, строки которой определяют имена пользователей, а столбцы - имена файлов. При этом файл считается доступным, если элемент таблицы не нулевой.

Метод шифрования состоит в зашифровки данных файла. Все пользователи имеют доступ к закодированным файлам, но только часть пользователей знает способ расшифровки данных. Файлы,требующие защиты, помечаются специальным признаком, называемом признаком защиты, который размещается в FAT.

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске и обеспечить совместное использование файлов несколькими пользователями и процессами. В широком смысле понятие «файловая система» включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие, например, как каталога файлов, дескрипторы файлов;
- таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

Имена файлов. Файлы идентифицируются именами, пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими, однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, WindowsNT в своей файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа. При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена файлам, имеющим длинные имена. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем. Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющим собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уникальное имя так, чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы.

Типы файлов. Файлы бывают разных типов:

- обычные файлы;
- специальные файлы;
- файлы-каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде (AmericanStandardCodeInformationInterchange). Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - это файлы, ассоциируемые с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Каталог - это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений, а с другой стороны - это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов различные параметры, например:

- информация о разрешенном доступе;
- пароль для доступа к файлу;
- владелец и создатель файла;
- признаки «только для чтения», «скрытый файл», «системный файл», «архивный файл» и др.;
- длина записи, ключа;
- времена создания, последнего доступа и последнего изменения;
- размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, или ссылаться на таблицы, содержащие эти характеристики. Каталоги могут образовывать иерархическую структуру за счет того, что каталог

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

более низкого уровня может входить в каталог более высокого уровня. Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог и сеть, если файл может входить сразу в несколько каталогов. В Windows каталоги образуют древовидную структуру, а в UNIX - сетевую. Как и любой другой файл, каталог имеет имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей Записи объединяются в файл, имя файла и его характеристики содержатся в специальной управляющей записи, называемой меткой файла. Записи не имеют имени и обращение к ним обеспечивается операционной системой по имени файла. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Операционная система обеспечивает программисту доступ к отдельной логической записи, объединяемых в файл.

Физическая организация и адрес файла. Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей - блоков, являющихся наименьшей единицей данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение - простейший вариант физической организации, при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока, другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация и пространство на диске исполь-

зуется не эффективно, так как отдельные участки маленького размера (минимально -один блок) могут остаться не используемыми.

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти. При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом - номером первого блока. В отличие от предыдущего способа каждый блок может быть присоединен в цепочку какого-либо файла, следовательно фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность организации доступа к произвольно заданному месту файла. Для того, чтобы прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки. Популярным способом, используемым, например, в файловой системе FAT, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент-индекс, располагающиеся в отдельной области диска (таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка.

Права доступа к файлу. Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список операций доступа, включающий следующие операции:

- создание и уничтожение файла;
- открытие и закрытие файла;
- чтение и запись в файл, поиск в файле

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- получение и установление новых значений атрибутов;
- переименование, выполнение файла и другие операции с файлами.

Кэширование диска. В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (например, диски), перехватываются промежуточным программным слоем - подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее часто используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

Глава 6. Языки программирования

6.1. Развития языков программирования

Язык программирования - формальная символьная система, предназначенная для описания алгоритмов с целью последующего выполнения в компьютере. Он характеризуется используемым алфавитом, синтаксическими и семантическими правилами для построения компьютерной программы. Процессор, выполняющий обработку информации в компьютере, представляет собой большую интегральную схему, в которой все команды и данные представлены в виде электрических сигналов, кодирующих совокупности нулей и единиц, представляющих собой различные коды команд и данных. Поэтому

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

на физическом уровне программное обеспечение, под управлением которого работает процессор, представляет собой последовательность, называемую машинным кодом, управляющим компьютером по определенному алгоритму. Описать даже простую последовательность действий компьютера в машинном коде весьма сложно. Поэтому для представления алгоритма в понятном компьютеру виде служат языки программирования.

Под программой понимается логически упорядоченная последовательность команд, необходимых для управления компьютером (выполнения им конкретных операций). Программирование же, как род деятельности, сводится к созданию последовательности команд, необходимой для решения определенной задачи, а люди, обученные процессу их составления (программированию), называются программистами. В результате алгоритм превращается в компьютерную программу. Однако лишь единственный язык программирования понятен компьютеру без переводчика - это язык его машинных команд (кодов). Команды, как и любая другая информация в машине, представляются в виде машинных двоичных кодов, которыми кодируются команды ЭВМ. Эти коды содержат две части, в одной из которых размещается код операции, а в другой - адреса исходных операндов и результата операции.

При использовании языков программирования процесс общения человека с машиной можно представить следующим образом (рис 6.1):

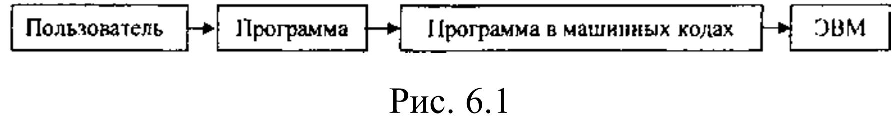


Рис. 6.1

Развитие языков программирования всегда осуществлялось и продолжает осуществляться в направлении исключения указанных недостатков. Первый шаг в этом направлении - это переход к языку Ассемблера. Отличие Ассемблера от языка машинных кодов заключается лишь в том, что двоичный код операции в Ассемблере заменяется на ее символьное имя. В результате для записи в программе команды "сложение" при использовании ас-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

семблера, например для сложения, пишется ее символическое имя "Add", а не конкретный код,например «0001». Однако при программировании на Ассемблере программисту также необходимо хорошее знание устройств компьютера. Эти недостатки исчезают при использовании языков высокого уровня, примерами которых являются BASIC, C, Pascal и многие другие языки. Большинство основных алгоритмических языков программирования созданы в 60-х и 70-х годах. За прошедший период времени периодически появлялись новые языки программирования, однако на практике они не получали широкого и продолжительного использования. Другим направлением в эволюции современных языков программирования были попытки создания универсальных языков (Алгол, PL/1, Ада), объединявших в себе достоинства ранее разработанных.

Появление персональных компьютеров и операционных систем с графическим интерфейсом привело к смещению внимания разработчиков программного обеспечения в сферу визуального или объектно-ориентированного программирования, баз данных. Это приводит к тому, что в настоящее время в качестве инструментальной среды используются конкретная среда программирования (Delphi, Access и др.) и можно в ряде случаев обходиться без знания базового языка программирования. Поэтому можно в определенном смысле полагать, что круг языков программирования будет постепенно стабилизироваться.

Анализ синтаксиса и семантики языков программирования показывает, что их идентичные конструкции различаются, главным образом, синтаксически, порядком следования компонентов и т.п. Семантическое же (смысловое) содержимое практически идентично за исключением небольших различий, не имеющих принципиального значения. Таким образом, конструкции современных языков имеют общее содержание (семантику), различный порядок следования компонент (синтаксис) и разные ключевые слова (лексику). Следовательно, различные языки подобного назначения предоставляют пользователю близкие возможности при различном внешнем виде программ.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Стандартизацию языков программирования в настоящее время осуществляют комитеты ISO/ANSI (Международный комитет стандартизации/Американский национальный институт стандартизации), однако их деятельность направлена в большей степени на неоправданное синтаксическое расширение языков. Для исключения существующих недостатков предложены способы задания семантического и синтаксического стандартов языков программирования.

Семантическое описание любой конструкции языка (оператора, типа данных, процедуры и т.п.) должно содержать не менее трех обязательных частей:

- список компонент языка;
- описание каждой компоненты;
- описание конструкции в целом.

Для синтаксического описания обычно используется специальный метаязык, например язык Бэкуса БНФ, такое описание имеет место в любом языке, начиная с Алгола.

Среди большого числа языков программирования заметную роль в развитии программирования сыграли три основных пары: Фортран и Алгол-60, Паскаль и Си, Java и Си++. Эти языки объединены не случайно в пары, так как «противостояние» заложенных в них идей способствовало дальнейшему развитию языков.

Трансляторы языков программирования. Языки программирования гораздо более понятны человеку, чем машинные коды, они не требуют знания устройства компьютера. Однако при их использовании, как и в случае Ассемблера, требуются специальные программы-переводчики с языка программирования на язык машинных кодов. Эти программы называются **трансляторами,** а процесс перевода - **трансляцией.** В результате схема общения человека с ЭВМ требует этапа трансляции для получения машинного кода. Важно различать язык программирования и его реализацию. Сам язык — это система записи, набор правил, определяющих синтаксис и семантику

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

программы. Реализация языка - это программа, которая преобразует запись высокого уровня в последовательность машинных кодов, Существуют два способа реализации трансляции с языка в машинные коды: компиляция и интерпретация и соответственно трансляторы двух типов: компиляторы и интерпретаторы

С помощью языка программирования создается еще не совсем готовая программа, а только ее текст, описывающий ранее разработанный алгоритм. Чтобы получить работающую программу, надо этот текст оттранслировать, т.е. либо автоматически перевести в машинный код (для этого служат программы-компиляторы) и затем использовать отдельно от исходного текста, либо сразу выполнять команды языка, указанные в тексте программы (это выполняют программы-интерпретаторы).

Интерпретатор берет очередной оператор языка из текста программы, анализирует его структуру и затем сразу исполняет (обычно после анализа оператор транслируется в некоторое промежуточное представление или даже машинный код для более эффективного дальнейшего исполнения). Только после того как текущий оператор успешно выполнен, интерпретатор перейдет к следующему. При этом, если один и тот же оператор должен выполняться в программе многократно, интерпретатор всякий раз будет выполнять его так, как будто встретил впервые. Вследствие этого, программы, в которых требуется осуществить большой объем повторяющихся вычислений, могут работать медленно. Кроме того, для выполнения такой программы на другом компьютере на нем также должен быть установлен интерпретатор - ведь без него текст программы является просто набором символов. По-другому, можно сказать, что интерпретатор моделирует некую виртуальную вычислительную машину, для которой базовыми инструкциями служат не элементарные команды процессора, а операторы языка программирования.

Компиляторы полностью обрабатывают весь текст программы (он называется исходным кодом). Они просматривают его в поисках синтаксических ошибок (иногда несколько раз), выполняют определенный смысло-

вой(семантический) анализ и затем автоматически переводят (транслируют) на машинный язык - генерируют машинный код. Нередко при этом выполняется оптимизация с помощью набора методов, позволяющих повысить быстродействие программы (например, с помощью инструкций, ориентированных на конкретный процессор, путем исключения ненужных команд, промежуточных вычислений и т.д.). В результате законченная программа получается компактной и эффективной, работает в сотни раз быстрее программы, выполняемой с помощью интерпретатора, и может быть перенесена на другие компьютеры с процессором, поддерживающим соответствующий машинный код.

В реальных системах программирования иногда совмещаются технологии компиляции и интерпретации. В процессе отладки программа может выполняться по шагам, а результирующий код не обязательно будет машинным.

С помощью языков низкого уровня типа ассемблера создаются очень эффективные и компактные программы, так как разработчик получает доступ ко всем возможностям процессора. С другой стороны, при этом требуется очень хорошо понимать устройство компьютера, затрудняется отладка больших приложений, а результирующая программа не может быть перенесена на компьютер с другим типом процессора. Подобные языки обычно применяют для написания небольших системных приложений, драйверов устройств, модулей стыковки с нестандартным оборудованием, когда важнейшими требованиями становятся компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам. В некоторых областях, например в машинной графике, на языке Ассемблера пишутся библиотеки, эффективно реализующие алгоритмы обработки изображений, требующие вычислений с высокой скоростью.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

6.2. Языки программирования высокого уровня

Язык программирования высокого уровня - это язык, разработанный

для удобного и быстрого описания программистом алгоритмов решаемых задач. Основная черта языков высокого уровня использование определенного уровня абстракций, то есть введение семантических конструкций, кратко описывающих используемые структуры данных и операции над ними, которые потребовали бы существенно более сложного описания на языках низкого языка, таких как Ассемблер. Это позволяет сократить значительно время программирования и последующей отладки, что особенно важно для сложных программ. Очень важным является то, что данный класс языков значительно ближе и понятнее человеку, чем компьютеру. Особенности конкретных компьютерных архитектур в них не учитываются, поэтому создаваемые программы на уровне исходных текстов легко переносимы на другие платформы, для которых создан транслятор этого языка. Разрабатывать программы на языках высокого уровня с помощью понятных и мощных команд значительно проще, а ошибок при создании программ допускается гораздо меньше. Однако следует также и отметить то, что программы на языках высокого уровня менее эффективны, чем программы на языках низкого уровня и поэтому большинство систем программирования на языках высокого уровня допускают вставку фрагментов на Ассемблерах в подобные программы.

Наиболее распространёнными языками высокого уровня являются C++, VisualBasic, Java, Delphi, PHP. Языкам высокого уровня свойственно умение работать с комплексными структурами данных. В большинство из них интегрирована поддержка строковых типов, объектов, операций файлового ввода-вывода и т. п. Рассмотрим кратко основные языки программирования высокого уровня.

Fortran. В 1954 году был создан первый язык программирования высокого уровня - Фортран (FORTRAN - FORmulaTRANslator), являющийся первым компилируемым языком. Программисты, разрабатывавшие программы исключительно на ассемблере, выражали серьезное сомнение в возможности появления высокопроизводительного языка высокого уровня, поэтому

основным критерием при разработке компиляторов Фортрана являлась эффективность исполняемого кода. Хотя в Фортране впервые был реализован ряд важнейших понятий программирования, удобство создания программ было принесено в жертву возможности получения эффективного машинного кода. Однако для этого языка было создано огромное количество библиотек, начиная от статистических комплексов и кончая пакетами управления спутниками, поэтому Фортран продолжает активно использоваться во многих организациях, в 2001 году завершились работы над очередным стандартом Фортрана. Имеется стандартная версия Фортрана HPF (HighPerformanceFortran) для параллельных суперкомпьютеров со множеством процессоров.

Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки удобны для человека для создания программ до нескольких тысяч строк длиной. Однако такой язык становился нечитаемым и трудно управляемым, когда дело касалось больших программ. Решение этой проблемы пришло после изобретения языков структурного программирования (structuredprogramminglanguage). Структурное программирование предполагает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций безусловного перехода (GOTO), автономные подпрограммы, поддержку рекурсий и локальных переменных. Основа данного подхода состоит в возможности разбиения программы на составляющие элементы.

Pascal.Язык структурного программирования, созданный в конце 70-х годов основоположником множества идей современного программирования

Н.Виртом, во многом напоминает Algol, но в нем ужесточен ряд требований к структуре программы и имеются возможности, позволяющие успешно применять его при создании крупных проектов.

С. Данный язык структурного программирования был создан в лаборатории Bell и первоначально не рассматривался как массовый. Он планировался для замены Ассемблера, чтобы иметь возможность создавать эффек-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

тивные и компактные программы и, в то же время, не зависеть от конкретного типа процессора. Язык С во многом похож на Паскаль и имеет дополнительные средства для прямой работы с памятью (указатели). На этом языке в 70-е годы написано множество прикладных и системных программ и ряд известных операционных систем, например Unix.

Basic. Для этого языка процедурного программирования имеются компиляторы и интерпретаторы, а по популярности он занимает первое место в мире. Он создавался в 60-х годах в качестве учебного языка и очень прост в изучении.

Хотя структурное программирование при его использовании дало выдающиеся результаты, даже оно оказывалось несостоятельным тогда, когда программа достигала определенной длины. Для того, чтобы написать более сложную и длинную программу, нужен был новый подход к программированию. В итоге в конце 1970-х годов были разработаны принципы объектно-ориентированного программирования. Оно сочетает в себе лучшие принципы структурного программирования с новыми мощными концепциями, базовые из которых называются инкапсуляцией, полиморфизмом и наследованием. Примерами объектно-ориентированных языков программирования являются: ObjectPascal, C++, Java и др. Объектно-ориентированное программирование позволяет оптимально организовывать программы, разбивая проблему на составные части и работая с каждой по отдельности. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть объектов, относящихся к этой задаче.

C++. Это объектно-ориентированное расширение языка С, созданное в 1980 году. Множество новых мощных возможностей, позволивших резко повысить производительность программистов, наложилось на унаследованную от языка С определенную низкоуровневость, в результате чего создание сложных и надежных программ потребовало от разработчиков высокого уровня профессиональной подготовки.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Java. Этот язык был создан компанией Sun в начале 90-х годов на основе С++. Он призван упростить разработку приложений путем исключения из него всех низкоуровневых возможностей. Но главная особенность этого языка - компиляция не в машинный код, а в платформно-независимый байт-код, который может выполняться с помощью интерпретатора виртуальной Java-Машины JVW (JavaVirtualMachine), версии которой созданы сегодня для многих популярных платформ. Благодаря наличию множества Java-машин программы на Java можно переносить не только на уровне исходных текстов, но и на уровне двоичного байт-кода, поэтому по популярности язык Java сегодня занимает второе место в мире после различных версий языка Бейсик.

Особое внимание в развитии этого языка уделяется двум направлениям: поддержке всевозможных мобильных устройств и микрокомпьютеров, встраиваемых в бытовую технику (технология Jini) и созданию платформо-независимых программных модулей, способных работать на серверах в глобальных и локальных сетях с различными операционными системами (технология JavaBeans). Пока основной недостаток этого языка - невысокое быстродействие, так как язык Java интерпретируемый. Резюмируя рассмотрение данного вопроса следует отметить, что из универсальных языков программирования сегодня наиболее популярными являются:

- Basic (Бейсик) - для освоения требует начальной подготовки (общеобразовательная школа);
- Pascal (Паскаль) и С(Си) - требует специальной подготовки (школы с углубленным изучением предмета и общетехнические высшие учебные заведения);
- С++ (Си++), Java (Ява) - требуют профессиональной подготовки (специализированные высшие учебные заведения).

Для каждого из этих языков программирования сегодня имеется немало систем программирования, выпускаемых различными фирмами и ориентированных на различные модели персональных компьютеров и операцион-

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

ные системы. Наиболее популярны следующие визуальные среды проектирования программ для операционной системы Windows:

- Basic: Microsoft Visual Basic;
- Pascal: Borland Delphi;
- C++: Borland C++Builder;
- Java: Symantec Cafe.

Для разработки серверных и распределенных приложений можно использовать систему программирования MicrosoftVisualC++, продукты фирмы Borland, практически любые средства программирования на Java.

6.3. Языки программирования контроллеров систем управления

Современные контроллерные системы управления для программирования используют более простые языки программирования, адаптированные к задачам, решаемым автоматизированными системами управления технологическими процессами. Для систематизации разработки программ для контроллеров логического управления Международная электротехническая комиссия утвердила стандарт IEC-1131, который включает 5 языков:

1) Язык инструкций (InstructionList - IL) - входной язык, аналогичный Ассемблеру. Написанная на нем программа представляет собой список последовательно выполняемых команд, которые адаптированы к задачам управления.

2) Релейно-контактная схема РКС (LadderDiagrams - LD) - графический язык программирования, который обеспечивает построение на экране монитора схемы, аналогичной принципиальной электрической схеме и с помощью специальных программ создается программа управления контроллерной системой. Такой подход к программированию отличается наглядностью при решении задач логического управления.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

3) Схема функциональных блоков (FunctionBlockDiagram - FBD) - графический язык программирования, при котором на экране программатора составляется схема, аналогичная принципиальной электрической схеме на логических элементах. Эти схемы позволяют изображать последовательность обработки сигналов как логических, так и цифровых в достаточно наглядном виде.

4) Последовательно-функциональная схема или Графсет (SequentialFunctionChart - SFC) - графический язык программирования, аналогичный блок -схеме алгоритма. Этот язык удобен для программирования процессов с последовательными операциями и наличием сложных ветвлений в алгоритмах управляющих программ. Стиль программирования на данном языке предполагает разработку программы «сверху вниз». Для программирования элементов самого низкого уровня необходимо использование одного из указанных ранее языков.

5) Язык структурированного текста (StructuredText - TS) - язык, аналогичный языку Pascal.

Программные приложения для систем управления в ряде случаев могут дополняться или полностью создаваться на основе языков программирования общего назначения, например в среде Delphi,Visual C или в пакетах систем управления базами данных.

6.3.1. Этапы разработки и функциональные спецификации

Появление промышленных компьютеров и контроллеров оказало глубокое воздействие на методологию проектирования цифровых систем управления различного назначения. Главной задачей проектирования становится разработка и реализация программных средств, то есть прикладных программ, которые отражают спецификации конкретного применения контроллеров и возможности его системы команд. До настоящего времени разработка прикладных программ в основном опирается на опыт и здравый смысл,

которым достаточно трудно научить, они приобретаются при разработке собственных программ и тщательном анализе разработанных программ. Основой успеха при разработке программного обеспечения является детальное планирование, четкая организация, тщательное и полное документирование.

Разработка прикладных программ для компьютерных систем управления включает в себя следующие основные этапы:

- анализ требований и разработка функциональных спецификаций;
- разделение задачи на части (модули) и выбор или разработка алгоритмов решения каждого из частей;
- программирование алгоритмов и транслирование программы;
- проверка (тестирование) и отладка программы;
- документирование и сопровождение программ.

Качество проектирования управляющей системы зависит от того, насколько хорошо разработчик понял и сформулировал решаемую проблему и определил рабочие характеристики системы. Общие функциональные спецификации микроконтроллерной системы обычно включают в себя:

- краткий раздел с четким описанием проблемы, решаемой системой;
- список необходимых аппаратных средств с определениями входных и выходных сигналов;
- описание связей программных модулей;
- полное описание системы и особенностей ее взаимодействия с входными и выходными устройствами;
- инструкцию для потенциальных пользователей, содержащую спецификации входных данных и выходных результатов, реакции на особые случаи и т. п.

На составление функциональных спецификаций следует отвести достаточное время. Практика показывает, что примерно две трети ошибок в прикладных программах появляются из-за недостаточно полных и четких спецификаций. Такие логические ошибки наиболее трудно локализовать и исправить. Оставшаяся треть приходится на ошибки кодирования, исправлять

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

которые обычно проще. Естественно, первоначальные спецификации никогда не будут настолько полными, чтобы раз и навсегда разрешить все вопросы проектирования. Со временем могут возникнуть непредвиденные обстоятельства и даже появиться изменения в технических требованиях. Поэтому рекомендуется планировать возможные изменения спецификаций и учитывать потенциальные возможности расширения аппаратных и программных средств системы. Разработка адекватных и полных спецификаций на самом раннем этапе проектирования сэкономит много времени и средств. На данном этапе целесообразно использовать графические представления решаемых задач в виде схем-алгоритмов, диаграмм, таблиц. Наконец, после оформления полных функциональных спецификаций следует строго их придерживаться, иначе можно «утонуть» в море улучшений и модификаций, появление которых характерно для неадекватных спецификаций.

Данный этап проектирования самый важный, так как неправильное формулирование требований приводит к выполнению ненужной работы, а недооценка сложности вызывает расход средств и времени. Сегодня около 60% крупных проектов завершаются неудачей именно из-за ошибок на стадии подготовки требований. На основе требований по различным методикам определяется примерный объем проекта и его трудоемкость, рассчитываются будущие трудозатраты и определяется его стоимость. Так как требования к проекту во время работы над ним могут уточняться и меняться, а выполнение требований надо отслеживать, применяются специальные программы для управления требованиями.

Часто заказчик не в состоянии точно сформулировать на начальной стадии все свои требования и пожелания и поэтому специалисты по системному анализу должны помочь ему выразить требования в виде, пригодном для формализации. После согласования требований подписывается контракт на разработку программного обеспечения. В дальнейшем любые отклонения от сформулированных требований к продукту (как со стороны заказчика, так и со стороны исполнителя) рассматриваются как нарушение контракта.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Каждый этап требует от заказчика вложения собственных трудозатрат и привлечения высокопрофессиональных специалистов, поэтому он обязательно должен оплачиваться. Однако, хотя первый этап самый важный, заказчик редко понимает эту важность и не готов платить достаточно большие суммы. Примерный объем работ на этом этапе - 5-10 % от объема всего проекта.

6.3.2. Разделение задачи на части и алгоритмизация

Большинство современных применений микроконтроллерных систем связано с решением довольно сложных проблем, которые в недалеком будущем должны еще более усложняться. Поэтому после составления функциональных спецификаций целесообразно разделить общую проблему на простые и управляемые части. Программная реализация каждой из частей называется блоком (модулем). Блоки могут быть сложными, например арифметические операции в формате с плавающей точкой или простыми, например, преобразование формата данных. Сложные блоки разделяются на меньшие блоки до такого уровня, чтобы разработка алгоритмов работы каждого из них стала достаточно простой и очевидной.

Основные блоки выделяются из функциональных спецификаций и содержат управляющий блок (основная программа), блок интерфейса с периферийными устройствами, блоки реакции на прерывания и блоки выполнения программных функций, например поиска данных, сортировки, математических и логических преобразований и т. п. Особое внимание уделяется организации ввода-вывода с учетом особенностей команд обмена данными, реакций на прерывания, способов идентификации прерывающих устройств. Интерфейс с быстродействующими периферийными устройствами организуются по способу прямого доступа к памяти.

Большоезначение имеет правильная организация данных. Следует задать и описать форматы входных и выходных величин, промежуточных и

окончательных результатов, возможные варианты упаковки данных, выбрать организацию данных в памяти. Целесообразно каким-либо образом упорядочить данные в виде таблиц, массивов, списков и т. п. Рациональная организация данных может сократить длину прикладной программы или уменьшить время ее выполнения.

Далее находятся приемлемые компромиссные решения между аппаратными и программными средствами. Например, умножение можно реализовать подпрограммой, которая выполняется сравнительно медленно, но занимает всего несколько байт программной памяти. Кроме того, для умножения можно применить специализированную БИС, которая формирует произведение сомножителей значительно быстрее подпрограммы. По возможности в разрабатываемую программу следует включать уже готовые и отлаженные подпрограммы.

После выделения функциональных блоков разрабатываются алгоритмы их выполнения. В принципе, алгоритмы составляются, безотносительно к конкретным языку и контроллеру, но уже здесь желательна ориентация на особенности контроллерной реализации. Логику алгоритмов удобно представлять графическими блок-схемами такого уровня, при котором отдельные их элементы соответствуют нескольким машинным командам. При разработке алгоритмов большое значение имеют опыт и квалификация программиста и поэтому можно дать только общие рекомендации, которые в основном, сводятся к следующему:

- подробно рассмотреть функциональное назначение блока;
- проанализировать источники данных, их форматы;
- рассмотреть необходимость предварительной обработки данных, например, масштабирование, переход к относительным единицам, упаковка и т. п.;
- алгоритмизировать те преобразования данных, которые должен выполнять блок с максимальным использованием готовых подпрограмм;

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- определить форматы выходных данных и рассмотреть сопряжение блока с получателями данных.

Разработка алгоритмов занимает при проектировании прикладных программ значительную часть времени. Она представляет собой итеративную процедуру, когда для достижения необходимого результата приходится делать несколько пробных попыток. Значительное внимание приходится уделять тем временным ограничениям, которые характерны для большинства прикладных программ. Во многих случаях производительность контроллерных систем можно существенно повысить за счет увеличения объема памяти. Например, вычисление многих функций традиционно реализуется путем разложения их в ряд с достаточным для требуемой точности числом членов. Другой, более быстродействующий способ заключается в применении таблиц, хранимых в памяти. В предельном случае таблицы составляются для каждого значения аргумента, а в более реальном - для некоторых значений аргументов (узлов) с линейной интерполяцией между ними.

6.3.3. Описание алгоритмов на языке программирования

Данный этап включает написание исходной программы в текстовом редакторе и трансляцию программы для получения объектного (машинного) кода. Так как текст программы записывается с помощью ключевых слов, обычно происходящих от слов английского языка и набора стандартных символов для записи различных операторов, создавать текст программы можно в любом редакторе, получая в итоге текстовый файл с исходным текстом программы. Однако лучше использовать специализированные редакторы, которые ориентированы на конкретный язык программирования и позволяют в процессе ввода текста выделять ключевые слова и идентификаторы разными цветами и шрифтами. Подобные редакторы созданы для всех популярных языков и дополнительно могут автоматически проверять правильность синтаксиса программы непосредственно во время ее ввода. Далее исходный

файл с помощью программы-компилятора переводится в двоичный файл, имеющий стандартное расширение «.OBJ» или «.HEX».

Текст большой программы состоит, как правило, из нескольких модулей (файлов с исходными текстами), потому что хранить все тексты в одном файле неудобно - в них сложно ориентироваться. Каждый модуль транслируется в отдельный файл, которые затем объединяются. Кроме того, к ним надо добавить машинный код подпрограмм, реализующих различные стандартные функции,например математические, тригонометрические и многие другие сложные функции. Такие функции содержатся в библиотеках (файлах со стандартным расширением .LIB), которые поставляются вместе с компилятором. Сгенерированный код модулей и подключенные к нему стандартные функции надо не просто объединить в одно целое, а выполнить такое объединение с учетом требований операционной системы, то есть получить на выходе программу, отвечающую определенному формату.

Объектный код обрабатывается специальной программой - редактором связей или сборщиком, который выполняет связывание объектных модулей и машинного кода стандартных функций, находя их в библиотеках, и формирует на выходе работоспособное приложение - исполнимый код для конкретной платформы.

Исполняемый код - это законченная программа, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась. Как правило, итоговый файл имеет расширение .EXE или.COM. Таким образом, для создания компьютерной программы нужны следующие средства программирования:

- текстовый редактор;
- компилятор;
- редактор связей;
- библиотеки функций.

Как правило, в стандартную поставку систем программирования входят обычно три последних компонента, но хорошая интегрированная система

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

включает в себя и специализированный текстовый редактор, причем почти все этапы создания программы в ней автоматизированы: после того как исходный текст введен, его компиляция и сборка выполняются одним нажатием клавиши. Это очень удобно, так как не требует ручной настройки множества параметров запуска компилятора и редактора связей, указания им нужных файлов вручную и т. д. Процесс компиляции обычно демонстрируется на экране: показывается, сколько строк исходного текста откомпилировано, или выдаются сообщения о найденных ошибках.

Следует обратить самое серьезное внимание на документирование созданной программы. Обычно программа начинается с заголовка-комментария, в котором кратко описываются основные характеристики программы: имя, дата составления, версия, требования к памяти и особенности ввода-вывода. После заголовка следуют директивы определения непосредственных данных, назначения адресов портам ввода, директивы резервирования памяти, таблицы переходов и подпрограммы ввода-вывода. Далее следует основная программа, представляющая собой последовательность инициализаций и вызовов подпрограмм. Подпрограммы размещаются с учетом их уровней вложения. Завершается программа таблицами данных, если они есть. Для наглядного представления размещения элементов программы рекомендуется построить так называемую карту памяти с выделением следующих областей:

- специальная память, определяемая особенностями контроллера. Например, во многих контроллерах первые несколько десятков ячеек адресного пространства зарезервированы для реакции на прерывания;
- рабочая память, назначаемая для стека, подпрограмм ввода-вывода, передачи параметров подпрограммам и буферных областей. Необходимо тщательно следить за тем, чтобы при выполнении программы не происходило нарушения границ областей. Особенно это относится к области стека, который автоматически используется некоторыми командами;

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- память портов ввода-вывода, выделяемая при использовании ввода-вывода, отображенного на память;
- резидентная программная память, которая назначается для системных программ.

6.3.4. Тестирование и отладка программы

Процесс поиска ошибок в программе называется **тестированием,** а процесс устранения ошибок - **отладкой,** то есть они заключаются в локализации и удалении ошибок из программы. Практика показывает, что на отладку программы часто уходит больше времени, чем на ее проектирование и кодирование. Анализируется, в частности, устойчивость работы программы при вводе недопустимых или критических значений, при неверных действиях, в критических режимах и т. п. Когда число ошибок, выявляемых за определенный срок (неделя, месяц), снижается ниже экспериментально подобранного уровня (на основе аналогичных проектов), начинается тестирование программы у заказчика. К такому тестированию привлекается максимально возможное число сотрудников, и программа уже начинает частично функционировать в рабочем режиме. Следует убедиться, что кодирование соответствует логике блок-схемы, что циклы правильно инициализируются и завершаются. Необходимо разработать контрольные тесты для проверки функционирования каждого модуля. В тестах предусматриваются наихудшие случаи и предельные значения параметров. Закончив отладку простейших подпрограмм, можно переходить к подпрограммам следующего уровнями,и так дальше до основной программы. Большое внимание следует уделить передаче подпрограммам параметров.

Обычно этап отладки реализуется на системе проектирования или с помощью программ-симуляторов на персональном компьютере. Завершается отладка функциональным тестом, проверяющим правильность работы программы в условиях эксплуатации. Симулятор представляют собой комплекс

программ, которые дают возможность отладить прикладную программу компьютерной системы управления на персональном компьютере до проектирования или параллельно с проектированием аппаратных средств. В их составе обычно предусматривают программу-редактор для ввода и модификации прикладных программ, транслятор и моделирующую (имитирующую) программу. В моделирующей программе все регистры и ячейки памяти контроллера являются доступными пользователю, и программа имитирует выполнение каждой команды прикладной программы с соответствующим преобразованием содержимого этих регистров и ячеек. С помощью команд, вводимых с клавиатуры (интерактивная отладка), пользователь может производить следующие основные действия:

- запускать и останавливать выполнение программы по любому заданному адресу (адресам). При останове выдается определенная информация: количество команд и время их выполнения, содержимое регистров и ячеек памяти и другие параметры;
- выполнять прикладную программу по командам с выводом содержимого внутренних регистров (это действие называется трассировкой или прослеживанием программы);
- выводить содержимое определенной области памяти;
- модифицировать содержимое любого регистра контроллера.

Отлаженная программа затем загружается в память контроллера.

6.4. Жизненный цикл и сопровождение программного обеспечения

Под жизненным циклом программного обеспечения понимают период разработки и эксплуатации программного обеспечения, в котором обычно выделяют следующие этапы:

- возникновение и исследование идеи;
- анализ требований и проектирование;
- программирование;

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- тестирование и отладка;
- ввод программы в действие;
- завершение эксплуатации.

Этапы жизненного цикла программы показывают, что надежность и сопровождение программного обеспечения тесно связаны между собой, так как высокая надежность программного обеспечения достигается при многократном повторении цикла.Под надежностью программного обеспечения понимают вероятность его работы без сбоев и отказов в течение определенного периода времени, рассчитанного с учетом стоимости для пользователя каждого отказа. Из данного определения можно сделать следующие важные выводы:

- надежность программного обеспечения является не только внутренним свойством программы;
- надежность программного обеспечения - это функция, как самих программ, так и действий их пользователей.

Основными причинами ошибок программного обеспечения являются: большая сложность программного обеспечения, например, по сравнению с аппаратурой компьютера. Источниками ошибок программного обеспечения являются:

- внутренние: ошибки проектирования, ошибки алгоритмизации, ошибки программирования, недостаточное качество средств защиты, ошибки в документации;
- внешние: ошибки пользователей, сбои и отказы аппаратуры ЭВМ, искажение информации в каналах связи, изменения конфигурации системы.

Важным этапом жизненного цикла программного обеспечения, определяющим качество и надёжность системы, является тестирование. Тестирование - процесс выполнения программ с намерением найти ошибки. Основные этапы тестирования:

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- автономное тестирование, контроль отдельного программного модуля отдельно от других модулей системы;
- тестирование в сопряжении, контроль сопряжения (связей) между частями системы (модулями, компонентами, подсистемами);
- тестирование функций, контроль выполнения системой автоматизируемых функций;
- комплексное тестирование, проверка соответствия системы требованиям пользователей;
- тестирование полноты и корректности документации, выполнение программы в строгом соответствии с инструкциями;
- тестирование конфигураций, проверка каждого конкретного варианта поставки (установки) системы.

На основании методов обнаружения ошибок можно разработать следующие средства повышения надёжности программного обеспечения.

1. Средства, использующие временную избыточность:

- анализ доступных пользователю ресурсов,
- выделение ресурсов согласно ролям и уровням подготовки пользователей,
- разграничение прав доступа пользователей к отдельным задачам.

2. Средства, использующие информационную избыточность:

- открытая система кодирования, позволяющая пользователю в любой момент изменять коды любых объектов классификации;
- механизмы проверки значений контрольных сумм записей системы, обеспечивающих выявление всех несанкционированных модификаций (ошибок, сбоев) информации;
- средства автоматического резервного копирования и восстановления данных (в начале, конце сеанса работы или по запросу пользователей) обеспечивающие создание на рабочей станции клиента копий про-

граммы, которая может быть использована в случае аварийного сбоя аппаратуры и перехода на локальный режим работы

3. Средства, использующие программную избыточность:

- распределение реализации одноименных функций по разным модулям программного обеспечения с использованием разных алгоритмов и системы накладываемых ограничений и возможностью сравнения полученных результатов;

- средства обнаружения и регистрации ошибок в сетевом и локальном протоколах, в программные модули системы встроены средства протоколирования процессов сложных расчётов с выдачей подробной диагностики ошибок.

Сопровождение программного обеспечения. Международный Комитет IEEE определяет термин "сопровождение программного обеспечения" как "процесс модификации программной системы или ее компонент, проводимый после поставки системы заказчику с целью устранения отказов, улучшения производительности или других атрибутов системы или адаптации к изменившемуся программному окружению". Сопровождение программного обеспечения представляет процесс, позволяющий уже существующим продуктам выполнять свои функции с продолжением продаж, установок и использования заказчиками. В общем смысле, под сопровождением понимают все работы, проводимые уже после выпуска первоначального продукта. Средства накопления сообщений об отказах, ошибках, предложениях на изменения, выполненных корректировках являются основной для сопровождения комплекса программ.

Сопровождение может быть функционально разделено на четыре направления:

- корректирующее сопровождение, включающее в себя диагноз и исправление ошибок;

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- адаптивное сопровождение, позволяющее программному обеспечению должным образом взаимодействовать с изменяющейся окружающей средой (аппаратные средства и программное обеспечение);
- сопровождение, направленное на добавление новых и изменение существующих функций, а также повышение общего уровня производительности;
- профилактическое сопровождение (все еще относительно редкое), модифицирующее программное обеспечение для повышения надежности.

Сопровождение программного обеспечения состоит из следующих видов работ.

В первую группу работ входят работы по поддержанию работоспособности и актуальности программного обеспечения:

- устранение нарушений в работе системы;
- тестирование и исправление программного обеспечения;
- поиск и устранение нарушений логической и физической структуры данных;
- архивирование и восстановление данных из архивов;
- своевременное обновление релизов (версий) программных продуктов;
- обновление форм отчетности.

Вторая группа включает работы по защите информации:

- защита от несанкционированного доступа к данным;
- ограничение прав пользователей на изменение информации;
- настройка интерфейса пользователей, меню инструментов;
- редактирование списка пользователей.

Третья группа включает работы по внедрению системы:

- начальное обучение пользователей;

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

- настройка системы на особенности предприятия;
- изменение любых документов и отчетов конфигурации в соответствии с пожеланиями заказчика;
- создание новых документов и отчетов;
- информационно-технологическое сопровождение.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Кнут Д. Искусство программирования - 3-е изд.- М.: Вильямс, 2007.

Кудрявцев А.С., Гладышев Н.Н., Короткова Т.Ю. Моделирование элементов и систем теплоэнергоснабжения промышленных предприятий: учебное пособие/СПбГТУРП. СПб., 2006.

Кудрявцев А.С. Программирование и основы алгоритмизации: учебно-методическое пособие/СПбГТУРП. СПб., 2007.

Пойа Д. Математика и правдоподобные рассуждения.- М.: Наука, 1975.

Информатика: базовый курс /С.В.Симонович и др. - СПб.: Питер, 2007.

Брауде Э. Технология разработки программного обеспечения.- СПб.: Питер, 2004.

Прокопов А.А., Татаринцев Н.И., Цирлин Л.А. Компьютерные технологии автоматизации: учебное пособие/ СПбГТУ «ЛЭТИ», СПб., 2001.

Прокопов А.А., Татаринцев Н.И.,Цирлин Л.А. Применение программируемых контроллеров для управления технологическим оборудованием: учебное пособие/СПбГТУ «ЛЭТИ», СПб., 2001.

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Оглавление

ПРЕДИСЛОВИЕ 3

Глава 1. Основные понятия алгоритмов 5

1.1. Определение и свойства алгоритма .... -
1.2. Алгоритмические системы. Общие правила построения алгоритмов 14

Глава 2. Описание алгоритмов 22

2.1. Способы записи алгоритмов .... -
2.2. Структура алгоритма. Понятие базовых алгоритмических структур 31
2.3. Описание линейных алгоритмов 35
2.4. Описание разветвляющихся алгоритмов 38
2.5. Описание циклических алгоритмов 45
2.6. Описание алгоритмов с вложенными циклами 55
2.7. Особенности описания алгоритмов, ... 57
состоящих из нескольких модулей 57

Глава 3. Вычислительные алгоритмы инженерных задач 60

3.1. Общие подходы к разработке алгоритмов .... -
3.2. Последовательное суммирование и последовательное умножение 64
3.3. Понятие итерационных алгоритмов, их описание 67
3.4. Алгоритмы обработки массивов 83

Глава 4. Алгоритмы управления технологическими процессами 89

4.1. Алгоритмы управления дискретными процессами 91
4.2. Алгоритмы управления непрерывными технологическими процессами 106

Глава 5. Организация структур данных и файлов 121

5.1. Организация структур данных .... -
5.2. Организация файловой структуры 129

Глава 6. Языки программирования 137

6.1. Развития языков программирования .... -
6.2. Языки программирования высокого уровня 143
6.3. Языки программирования контроллеров систем управления 147
6.4. Жизненный цикл и сопровождение программного обеспечения 157

Библиографический список 162

НАУЧНО-ИНФОРМАЦИОННЫЙ ЦЕНТР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ТЕХНОЛОГИЧЕСКОГО УНИВЕРСИТЕТА РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Учебное издание

Валерий Николаевич Суриков
Александр Сергеевич Кудрявцев
Геннадий Алексеевич Петров
Евгений Васильевич Хардиков

ОСНОВЫ АЛГОРИТМИЗАЦИИ ИНЖЕНЕРНЫХ ЗАДАЧ

Учебное пособие

Редактор и корректор Т.А. Смирнова
Технический редактор Л.Я. Титова Темплан 2012, поз. 112
Подп. к печати 25.12.12.Формат 60x84/16. Бумага тип.№1. Печать офсетная.
Печ. л. 10,0. Уч.-изд. л. 10,0. Тираж 150 экз. Изд. №112. Цена «С». Заказ

ризограф Санкт-Петербургского государственного технологического университета растительных полимеров, 198095, СПб., Ул. Ивана Черных, 4.