Содержание

Тема 8. Алгоритмизация и программирование

8.1 Раздел теории

Алгоритмомназывается точное предписание, задающее преобразование исходных данных в искомый результат. При этом могут производиться числовые вычисления, преобразования буквенных выражений, некоторых символов и т.д.

Важнейшими свойствами алгоритма являются:

- определенность (детерминированность). В алгоритме должна соблюдаться однозначность предписываемой последовательности действий, не допускающая произвольного ее толкования;
- массовость. Алгоритм решения задачи разрабатывается в общем виде так, чтобы его можно было применить для решения задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, называемой областью применимости алгоритма;
- дискретность. Алгоритм должен быть разделен на отдельные элементарные акты;
- результативность (конечность). Алгоритм должен приводить к решению задачи за конечное число шагов;
- инвариантность по отношению к исполнителю. Алгоритм может оставаться неизменным при выполнении предписываемых им команд человеком или машиной любого типа.

Для решения любой относительно сложной задачи, как правило, могут быть разработаны несколько различных алгоритмов. Естественно, что следует выбрать лучший из них. Но когда мы говорим лучший, то всегда возникает вопрос: «С какой точки зрения лучший?»

Ниже представлены основные критерии качества алгоритма.

- Связанность алгоритма определяется количеством промежуточных результатов, которые должны одновременно храниться в памяти. Естественно, что алгоритм тем лучше, чем его связность меньше, так как при этом уменьшается количество ячеек, занятых в оперативной памяти.
- Объем алгоритма

Объем алгоритма – это количество операций (шагов), которые необходимо выполнить для получения конечного результата.

Чем меньше трудоемкость, т.е. чем меньше операций нужно предусмотреть на его написание и исполнение, тем выше качество алгоритма. Уменьшение количества шагов экономит не только время математика – составителя алгоритма, но и машинное время, сокращает длительность решения задачи на ЭВМ.

- Длительность решения определяется количеством шагов алгоритма, а также сложностью этих шагов. Даже если все операции, как в примере выше, ограничиваются четырьмя арифметическими действиями, то все равно сказывается существенная разница во время их выполнения. Умножение и деление требуют затрат машинного времени в 3–4 раза больше, чем простейшие операции – сложение и вычитание
- Разветвленность алгоритма характеризует логическую сложность и определяется количеством путей, по которым может реализовываться процесс вычислений. Значительная разветвленность увеличивает сложность алгоритма, а значит и трудоемкость его разработки.
- Цикличность алгоритма заключается в том, что фактическое количество операций, которые должны быть выполнены в ходе вычислительного процесса, превышает количество операций, содержащихся в записи алгоритма. (Те или иные операции могут повторяться многократно – в цикле).

Ниже представлено описание графического представление алгоритмов.

Схема алгоритма – это ориентированный граф, указывающий порядок исполнения алгоритма.

Схема алгоритма содержит условные графические фигуры. Они обозначают соответствующие команды, а соединяющие их линии указывают последовательность реализации этих операторов. Схема дает наиболее наглядное представление о структуре алгоритма.

При изображении алгоритмов в виде схем используют фигуры, изображенные на рисунке 8.1.

Рисунок 8.1 – Фигуры, предназначенные для изображения схем алгоритмов

Форма и размеры фигур регламентируются ГОСТ 19.701–90 «Схемы алгоритмов, программ, данных и систем.Условные обозначения и правила их выполнения».

Соотношения между геометрическими элементами устанавливаются следующие:

1. высота a выбирается из ряда 10, 15, 20 мм (допускается увеличивать размер на число, кратное 5);
2. ширина $b = 1{,}5a$.

Ниже описаны технологии программирования.

1. **Операционный подход**

   На начальных этапах развития вычислительной техники, когда машинное время было дорого, а возможности ЭВМ малы основными требованиями к алгоритму и программе были:

   1. использование наименьшего возможного числа ячеек оперативной памяти компьютера при исполнении программы;
   2. минимальное время исполнения (минимальное число операций).

   При этом программы составлялись из следующих команд:

   a. операции присваивания;
   b. простейших арифметических операций;
   c. операций сравнения чисел;
   d. операторов безусловного и условного переходов;
   e. операторов вызова подпрограмм.

   К недостаткам алгоритмов, построенных по операционному подходу:

   a. злоупотребление командой условного и безусловного переходов, приводящее к очень запутанной структуре программы;
   b. разнообразные уловки, направленные на повышение эффективности программы, приводили к ее непонятности, ненадежности, трудностям в отладке и модификации, делая программирование трудоемким, сложным и чрезвычайно дорогостоящим.
2. **Структурный подход**

   В основе структурного подхода лежит утверждение о том, что логическая структура любого алгоритма может быть выражена комбинацией трех канонических (базовых) структур (см. рисунок 8.2).

   a. Следование (последовательность) предписывает выполнение указанного набора действий в естественном порядке (один за другим) без пропусков и повторений.
   b. Выбор (ветвление) организует выполнение лишь одного из двух указанных действий в зависимости от справедливости некоторого условия.
   c. Цикл организует многократное выполнение указанного действия. Используется несколько различных форм записи циклов.

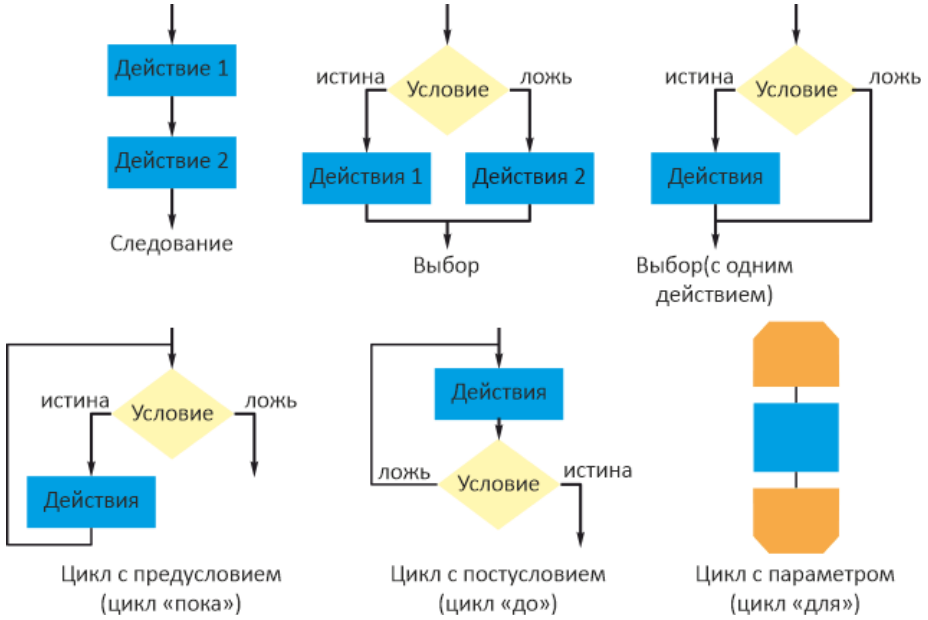

Рисунок 8.2 – Базовые структуры для изображения алгоритма

Еще одним важным компонентом структурного подхода является модульность.

Модуль – это последовательность логически связанных операций, оформленных как отдельная часть программы.

К преимуществам структурного подхода относятся:

a. возможность создания программы несколькими программистами;
b. простота проектирования и последующих модификаций;
c. упрощение отладки программы – поиска и устранения в ней ошибок;
d. возможность использования готовых библиотек наиболее употребительных модулей.

3. **Объектно–ориентированное программирование**

Объектно–ориентированное программирование основано на концепции объединения данных и процедур их обработки в единое целое.

Объект – это совокупность свойств (структур данных, характерных для этого объекта), методов их обработки (подпрограммы изменения свойств) и событий, на которые данных объект может реагировать и которые приводят, как правило, к изменению свойств объекта.

Свойства – это перечень параметров объекта, которые определяют внешний вид и поведение объекта, выделяют уникальные особенности каждого экземпляра.

События – это сигналы, формируемые пользователем, операционной системой или работающей программой.

Объекты могут иметь идентичную структуру и отличаться только значениями свойств. В таких случаях в программе создается новый тип, основанный на единой структуре объекта. Он называется классом, а каждый конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Основными отличительными свойствами объекта являются:

a. инкапсуляция – объединение структур (записей) с методами (процедурами и функциями), работающими с этими записями;
b. наследование – задание объекта, затем использование его для построения иерархии порожденных объектов с наследованием доступа каждого из порожденных объектов к коду и данным предка;
c. полиморфизм – задание одного имени действию, которое передается вверх и вниз по иерархии объектов, с реализацией этого действия способом, соответствующим каждому объекту в иерархии.

4. **Декларативный подход в программировании**

Появился в разработке компьютерных программ в начале 70–х. Направлен на относительно узкий круг задач искусственного интеллекта. При его применении программист описывает свойства исходных данных, их взаимосвязи, свойства, которыми должен обладать результат, а не алгоритм получения результата. Алгоритм порождается той системой, которая поддерживает декларативно–ориентированный язык программирования (Пролог и Лисп).

5. **Процедурно–ориентированное программирование**

Данный подход иногда называют программированием. В привычных алгоритмах и программах действия совершаются последовательно одно за другим. Однако логика решения множества задач вполне допускает одновременное выполнение нескольких операций, что ведет к многократному увеличению эффективности. Реализация параллельных алгоритмов на ЭВМ стала возможной с появлением многопроцессорных компьютеров.

Язык программирования – это формальный язык для описания алгоритма решения задачи на компьютере.

Каждый язык программирования имеет:

- алфавит
- синтаксис
- семантику

Алфавит – это фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.

Синтаксис – это система правил, определяющих допустимые конструкции языка программирования из букв алфавита;

Семантика – это система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

Программа, подготовленная на языке программирования, должна пройти ряд преобразований, чтобы компьютер мог ее выполнить (см. рисунок 8.3).

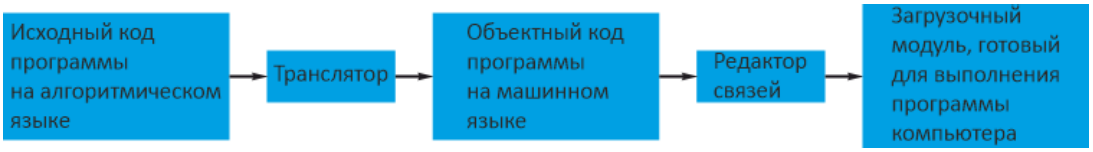

Рисунок 8.3 – Схема процесса создания загрузочного модуля программы

Трансляция может выполняться с использованием средств компиляторов или интерпретаторов. Компиляторы транслируют всю программу, но без ее выполнения. Интерпретаторы, в отличие от компиляторов, выполняют пооператорную обработку и выполнение программы.

Достоинством компилятора являются компактный и эффективный код программы и высокая скорость выполнения. Достоинством интерпретаторов является возможность постоянного контроля состояния программно–аппаратной среды, благодаря чему достигается высокая надежность работы.

В реальных системах программирования перемешаны технология и компиляции и интерпретации. В процессе отладки программа может выполняться по шагам, а для отлаженной программы может быть получен исполняемый код.

В зависимости от способа преобразования операндов языка программирования в инструкции для ЭВМ их подразделяют на:

- языки низкого уровня (трансляторы), которые осуществляют транслитерацию, т.е. преобразование одного оператора языка в одну машинную инструкцию (пример – ассемблеры)
- языки высокого уровня (компиляторы), осуществляющие преобразование одного оператора языка в несколько машинных инструкций.

Языки ассемблера ориентированы на конкретный тип процессора и учитывают его особенности, что позволяет создавать очень эффективные и компактные программы. Однако от разработчика в этом случае требуется очень высокая квалификация, отладка больших программ затруднена, а результирующая программа не может быть перенесена на другой тип компьютера.

Языки высокого уровня имитируют естественные языки и имеют следующие достоинства:

- алфавит значительно шире машинного, что делает его гораздо более выразительным и существенно повышает наглядность и понятность текста;
- набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- конструкции команд (операторов) отражают содержательные виды обработки данных и задаются в удобном для человека виде;
- используется аппарат переменных и действия с ними;
- поддерживается широкий набор типов данных.

Несмотря на значительные различия между языками программирования, ряд фундаментальных понятий в большинстве из них схожи.

Оператор – это законченная фраза языка, определяющая однозначно трактуемый этап обработки данных.

В теории алгоритмов выделяют основные (базисные) операторы языка: присвоения, условный и безусловный переход, пустой оператор. К производным, неосновным, относят: составной оператор, оператор выбора, оператор цикла и оператор присоединения. Все операторы в тексте программы отделяются друг от друга явными или неявными разделителями.

Большая часть операторов ведет обработку величин. Величины могут быть постоянными и переменными. Величины характеризуются типом, именем и значением. Наиболее распространенные типы величин – числовые (целые и вещественные), символьные и логические.

Величины классифицируются на простые и структурированные. Простая величина в каждый момент времени может иметь не более одного значения. Ей соответствует одна ячейка памяти компьютера. Структурированная величина, имея одно имя, может иметь сразу несколько значений. Эти значения представляют собой элементы (компоненты) величины.

Всем программным объектам даются индивидуальные имена. Имя программного объекта называют идентификатором. Чаще всего идентификатором является любая конечная последовательность букв и цифр, начинающаяся с буквы.

Описания или объявления программных объектов необходимы для определения их свойств. В некоторых языках стандартные описания простых числовых и символьных данных опускают (описание по умолчанию) или в них задаются правила описания по имени объекта. Особый интерес представляют в языках программирования описания нестандартных структур данных, таких как запись, файл, объект, список, дерево и т.п.

Переменная – это программный объект, способный принимать некоторое значение с помощью оператора присваивания. Каждая переменная после ее описания отождествляется с некоторой ячейкой памяти, содержание которой является ее значением.

Функция – это программный объект, задающий вычислительную процедуру определения значения, зависимого от некоторых аргументов. В каждом языке программирования имеется в наличии библиотека стандартных функций. Кроме того, программист может сам задавать новые функции.

Процедура – это программный объект, представляющий некоторый самостоятельный этап обработки данных. Процедура имеет входные и выходные параметры, называемые формальными. При использовании процедуры формальные параметры заменяются на фактические.

Модуль (Unit) – это специальная программная единица, предназначенная для создания библиотек и разделения больших программ на логически связанные блоки.

Современный подход к проектированию программ основан на декомпозиции задачи. Целью декомпозиции является создание модулей, которые представляют собой небольшие, относительно самостоятельные программы, взаимодействующие друг с другом по хорошо определенным и простым правилам.

Разработка любой программы или прикладной системы начинается с определения требований к ней для конкретного набора пользователей и заканчивается эксплуатацией системы этими пользователями.

По современным взглядам проектирование и разработку программ целесообразно разбить на ряд последовательных этапов:

1. постановка задачи;
2. проектирование программы;
3. построение модели;
4. разработка алгоритма;
5. реализация алгоритма;
6. анализ алгоритма и его сложности;
7. тестирование программы;
8. документирование.

При постановке задачи для крупных компьютерных программ необходимо провести следующие работы:

- выработать требования (свойства, качества и возможности), необходимые для решения проблемы или достижения цели (как правило, эта деятельность носит экспертный характер);
- разработать спецификации, включающие:
- цель программы;
- граничные условия;
- описание функций системы;
- спецификации входных и выходных данных;
- верификационные требования (установление тестовых случаев);
- тип и количество документов.

Проектирование программы осуществляется следующим образом. Сначала производится проектирование архитектуры программной системы. Это предполагает первичную (общую) стадию проектирования и заканчивается декомпозицией спецификаций в структуру системы. Обычно на модульном уровне разрабатывается спецификация каждого модуля:

- имя/цель – дается имя модулю и предложение о его функции с формальными параметрами;
- неформальное описание – обзор действий модуля;
- ссылки – какие модули ссылаются на него и на какие модули ссылается данный модуль;
- вход/выход – формальные и фактические параметры, глобальные, локальные и связанные (общие для ряда модулей) переменные;
- примечания – полезные комментарии общего характера по модулю.

Следующим шагом является детальное проектирование. На этом этапе происходит процедурное описание программы, выбор и оценка алгоритма для реализации каждого модуля. Входной информацией для проектирования являются требования и спецификации системы.

Построение модели в большинстве случаев является непростой задачей. При построении моделей, как правило, используют два принципа: дедуктивный (от общего к частному) и индуктивный (от частного к общему). При дедуктивном подходе рассматривается частный случай общеизвестной фундаментальной модели. При заданных предположениях известная модель приспосабливается к условиям моделируемого объекта.

Индуктивный способ предполагает выдвижение гипотез, декомпозицию сложного объекта, анализ, затем синтез. Здесь широко используется подобие, аналогичное моделирование, умозаключение с целью формирования каких–либо закономерностей в виде предположений о поведении системы.

Технология построения модели при индуктивном способе:

1. эмпирический этап (умозаключение, интуиция, предположение, гипотеза);
2. постановка задачи для моделирования;
3. оценка, количественное и качественное описание;
4. построение модели.

Разработка алгоритма – самый сложный и трудоемкий процесс, но и самый интересный в творческом отношении. Выбор метода разработки зависит от постановки задачи, ее модели. На этом этапе необходимо провести анализ правильности алгоритма, что очень непросто и трудоемко.

Наиболее распространенная процедура доказательства правильности алгоритма – это прогон его на множестве различных тестов. Однако это не гарантирует того, что не может существовать случая, в котором программа «не сработает».На этапе реализации алгоритма происходят конструирование и реализация алгоритма, включающие кодирование, интеграцию, тестирование (сертификацию).

По сути, проводится перевод проекта в форму программы для конкретного компьютера, сборка системы и ее прогон при тестовых и нормальных условиях для подтверждения ее работы в соответствии со спецификациями системы.

Анализ алгоритма и его сложности необходим для оценки ресурсов компьютеров, на которых он будет работать, времени обработки конкретных данных, приспособления в работе в локальных сетях и телекоммуникациях. Хотелось бы также иметь для данной задачи количественный критерий для сравнения нескольких алгоритмов с целью выбора более простого и эффективного среди них.

Перед началом эксплуатации программы необходим этап ее отладки и тестирования.

Тестирование – это процесс исполнения программ с целью выявления (обнаружения) ошибок.

Тестирование – процесс деструктивный, поэтому считается, что тест удачный, если обнаружена ошибка. Существуют различные способы тестирования программ:

- тестирование программы как «черного ящика» – стратегия «черного ящика» определяет тестирование с анализом входных данных и результатов работы программы.
- тестирование программы как «белого ящика» заключается в стратегии управления логикой программы, позволяет использовать ее внутреннюю структуру.

Различают альфа– и бета–тестирование, производимое соответственно специалистами разработчика и заказчика программного изделия.

Основные типы ошибок, встречающихся при программировании:

- обращения к переменным, значения которым не присвоены или не инициализированы;
- выход индексов за границы массивов;
- несоответствие типов или атрибутов переменных величин;
- явные или неявные проблемы адресации памяти;
- ошибочные передачи управления;
- логические ошибки.

Есть золотое правило программистов – оформляй свои программы в том виде, в каком бы ты хотел видеть программы, написанные другими. К каждому конечному программному продукту необходимо документированное сопровождение в виде помощи (*help*), файлового текста (*readme.txt*).