## § 8. ПРИНЦИПЫ РАЗРАБОТКИ АЛГОРИТМОВ И ПРОГРАММ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

### 8.1. ОПЕРАЦИОНАЛЬНЫЙ ПОДХОД

В настоящее время создание алгоритмов - написание программ для электронных вычислительных машин - стало видом человеческой деятельности. Важнейший конструктивный компонент программирования, не зависящий от особенностей синтаксиса языков программирования и специфики функционирования конкретных вычислительных машин, - разработка алгоритма.

Подходы к созданию алгоритмов и требования к ним существенно изменялись в ходе эволюции компьютеров. Первоначально, в эпоху ЭВМ 1 -го и 2-го поколений, когда они были еще мало распространены, машинное время было дорого, а возможности ЭВМ очень скромны (с точки зрения сегодняшних достижений), основным требованием к алгоритму была его узко понимаемая эффективность:

1) минимальные требования в отношении оперативной памяти компьютера -.программа должна была использовать наименьшее возможное число ячеек оперативной памяти компьютера;

2) минимальное время исполнения (минимальное число операций). При этом программы составлялись из команд, непосредственно или почти непосредственно исполнявшихся компьютером (точнее говоря, процессором):

• операции присваивания;

• простейших арифметических операций;

• операций сравнения чисел;

• операторов безусловного и условных переходов (изменяющих порядок вычисления команд в программе);

• операторов вызова подпрограмм (вспомогательных алгоритмов).

Такой подход в программировании (создании алгоритмов), ориентированный на непосредственно выполняемые компьютером операции, можно назвать операциональным.

Рассмотрим подробнее операции, которые выполняются компьютером и которые являются шагами программы при операциональном подходе.

Операция присваивания состоит в том, что некоторое значение фигурирующей в программе величины помещается в ячейку памяти компьютера. Эта ячейка может либо принадлежать оперативной памяти, либо находиться в арифметико-логическом устройстве, выполняющем основные операции (это устройство - часть процессора). После операции присваивания указанное значение сохраняется в ячейке памяти, куда оно было помещено, пока не будет заменено другим в результате другого присваивания. Ячейка памяти, где размещается значение, в программе обозначается именем (идентификатором) соответствующей переменной. Примеры идентификаторов: *а, х, у1, у2*. Важно помнить, что переменные и, соответственно, их значения могут быть разных типов - числовые (целые или действительные), литерные или логические. Значения различных типов представляются (кодируются) в компьютере по-разному, поэтому они должны соответствовать типам переменных, которым они присваиваются. При разработке алгоритма следует всегда помнить и 'тщательно различать типы переменных.

Набор простейших арифметических операций «сложения» (+), «вычитания» (-), «умножения» (\*) и «деления» (/) (причем во многих случаях следует тщательно отличать деление, выполняемое над целыми числами - в этом случае операция деления распадается на деление нацело и вычисление остатка от деления) позволяет записывать арифметические выражения с использованием числовых констант и идентификаторов переменных. Для определения порядка операций в выражениях чаще всего используют стандартное математическое соглашение о старшинстве операции, согласно которому старшими и выполняемыми в 1-ю очередь являются умножение и деление, а младшими - сложение и вычитание. Для изменения «естественного» порядка выполняемых операций служат скобки. Сравните, например, порядок операций в выражениях:

*(а* + 2) \* *х* и а + 2 \* *х.*

Что же касается порядка выполнения операций одного старшинства, то они, как правило, выполняются в порядке записи в выражении.

Операция сравнения числовых значений фактически сводится к определению знака разности этих значений. Этот знак отображается с помощью специальной ячейки памяти (флага знака результата) вычислительного устройства компьютера и может использоваться при выполнении условных переходов между командами (шагами) алгоритма.

Чтобы понять, что такое условные и безусловные переходы при выполнении алгоритма, надо исходить из того, что шаги или команды алгоритма обладают метками или адресами, и, помимо естественного порядка выполнения команд соответственно их записи, возможен и другой порядок, при котором последовательность выполнения команд определяется переходами на команды с определенными метками или адресами. **Безусловным** называется переход, для которого изменение порядка выполнения команд раз и навсегда определено и не зависит ни от каких условий. Условным называется переход, для которого порядок выполнения команд определяется по некоторому условию, чаще всего условию сравнения величин числовых типов.

Операция вызова подпрограммы (вспомогательного алгоритма) - это такой переход в последовательности команд алгоритма, при котором на определенном этапе выполнения алгоритма происходит вначале переход на другую программу (подпрограмму по отношению к той, откуда совершен переход), а затем, после ее завершения, возврат в точку вызова подпрограммы и продолжение выполнения команд, начиная со следующей после команды вызова подпрограммы, в-их естественном порядке. Очевидно, что операция вызова подпрограммы представляет собой переход, при котором запоминается адрес команды, следующей за командой вызова подпрограммы, что позволяет вернуться к исходному алгоритму (головной программе) после выполнения вспомогательного алгоритма (подпрограммы).

Отметим, что универсальность существующих компьютеров достигается за счет определенного набора команд, типа только что описанного, и автоматического механизма их выполнения, а проблема решения задачи с помощью компьютера состоит лишь в преобразовании решаемой задачи в последовательность этих команд. В качестве примера рассмотрим операциональное представление алгоритма вычисления квадратного корня из положительного числа *а* с помощью рекуррентной формулы:



n = 0,1,2,...

Можно показать, что .

Будем обозначать через *x0* нулевое приближение (за него в данном случае можно принять любое положительное число), через *ε* заданную точность вычислений и через *c0* какое-нибудь число, удовлетворяющее условию 0 < *c0* <  , необходимое для оценки достигнутой точности с помощью неравенства



Алгоритм вычисления *.*

1) ввести числа *а, ε, x0, c0;*

2) присвоить переменной *х* значение *у,*

3) присвоить переменной *у* значение *а/х;*

4) присвоить переменной *у* значение *х* + *у,*

5) присвоить переменной *х* значение *у/2;*

6) присвоить переменной *у* значение *x*2;

7) присвоить переменной *у* значение *y-а;*

8) присвоить переменной *у* значение *у/c0;*

*9)* присвоить переменной *δ* значение *у/2;*

10) сравнить *δ* и *ε*; если *δ > ε,* то перейти к команде 3, иначе перейти к следующей команде;

11) вывести числа *х, а* и *ε*;

12) стоп.

В этом алгоритме все команды, кроме 10, предполагают переход к следующим за ними по записи командам, и лишь команда 10, являющаяся командой условного перехода, меняет порядок исполнения команд - после нее в нарушение порядка может выполняться команда 3, т.е. она определяет циклическую конструкцию в алгоритме.

Поясним эту программу. Команда 2 помещает значение начального приближения *x0* в ячейку памяти, в которой хранятся значения переменной *х* (на каждом этапе вычислений в этой ячейке хранится значение *х,* равное значению одного из членов рекуррентной последовательности *xn*).

Команды 3-5 вычисляют по числу *х* число *(х* + *а/х)/2.* Результат помещается в ячейку памяти, в которой хранится значение переменной *х,* при этом старое значение «стирается» новым. Команды 6-9 вычисляют величину

,

с помощью которой оценивается сверху разность *х -*  . Важное значение имеет команда 10. По ней не производятся вычисления, а сравниваются между собой вычисленное значение 5 и заданная точность *ε.* Если *δ* > *ε*, то управляющее устройство вернет вычислительный процесс к команде 3 и заставит повторить процесс.

В противном случае, когда требуемая точность достигнута, печатается полученный результат и работа прекращается.

Данный алгоритм весьма экономичен: в качестве рабочих он использует всего две ячейки памяти (для переменных *х* и *у),* его команды так продуманы, что никакие операции не выполняются с избыточным повторением.

В данном примере не были использованы какие-либо специальные обозначения команд, чтобы сделать их независимыми от языка конкретных ЭВМ (такие языки называют Ассемблерами), чтобы стал ясен общий характер операционального подхода к разработке алгоритмов. Однако, ориентированность этого подхода на возможности и особенности ЭВМ с появлением большого числа компьютеров 3-го и особенно 4-го поколений не позволяла перейти к массовому промышленному программированию и стала сдерживать развитие вычислительной техники. Отметим основные недостатки алгоритмов, к которым приводил операциональный подход:

• злоупотребление командой условного и безусловного переходов зачастую приводило к очень запутанной структуре программы, напоминавшей по образному сравнению «блюдо спагетти»;

• вместе с разнообразными уловками, направленными на повышение эффективности программы (т.е. минимальных требований к оперативной памяти и минимального времени выполнения), это приводило к непонятности программ, их ненадежности, трудностям в отладке и модификации, делая программирование трудоемким, сложным и чрезвычайно дорогостоящим.

Необходимость ориентироваться на ограниченный набор команд компьютера, на его скромные возможности приводила к огромной трудоемкости, к сложности программ, к проблемам, связанным с ошибками в них. В результате узким местом в развитии вычислительной техники оказалось именно программирование.

### 8.2. СТРУКТУРНЫЙ ПОДХОД

С появлением массовых ЭВМ 3-го поколения устаревшая технология программирования оказалась основным фактором, сдерживающим развитие и распространение компьютерных (информационных) технологий, что подтолкнуло ведущие в этой сфере деятельности фирмы, в первую очередь IBM, к разработке новых методологий программирования. Появившийся в начале 1970-х годов новый подход к разработке алгоритмов получил название структурного.

С появлением структурного программирования описанные выше трудности были во многом преодолены. В основе технологических принципов структурного программирования лежит утверждение о том, что логическая структура программы может быть выражена комбинацией трех базовых структур: следования, ветвления и цикла (это содержание теоремы Бема-Якопини).

**Следование** - самая важная из структур. Она означает, что действия могут быть выполнены друг за другом, рис. 1.19:



*Рис. 1.19.* Структура «следование»

Эти прямоугольники могут представлять как одну единственную команду, так и множество операторов, необходимых для выполнения сложной обработки данных.

**Ветвление** - это структура, обеспечивающая выбор между двумя альтернативами. Выполняется проверка, а затем выбирается один из путей (рис. 1.20).

Эта структура называется также «ЕСЛИ - ТО - ИНАЧЕ», или «развилка». Каждый из путей (ТО или ИНАЧЕ) ведет к общей точке слияния, так что выполнение программы продолжается независимо от того, какой путь был выбран.



*Рис. 1.20.* Структура «ветвление»

Может оказаться, что для одного из результатов проверки ничего предпринимать не надо. В этом случае можно применять только один обрабатывающий блок, рис.1.21:



*Рис. 1.21.* Структура «неполное ветвление»

Цикл (или повторение) предусматривает повторное выполнение некоторого Набора команд программы. Если бы циклы не существовали, вряд ли занятие программированием было бы оправданным: циклы позволяют записать длинные последовательности операций обработки данных с помощью небольшого числа повторяющихся команд. Разновидности цикла изображены на рис. 1.22 и рис. 1.23.

Цикл начинается с проверки логического выражения. Если оно истинно, то выполняется «*a*», затем все повторяется снова, пока логическое выражение сохраняет значение «истина». Как только оно становится ложным, выполнение операций «а» прекращается и управление передается по программе дальше.



*Рис. 1.22.* Структура цикла «пока»



*Рис. 1.23.* Структура цикла «до»



*Рис. 1.24.* Нахождение суммы трех чисел



*Рис. 1.25.* Нахождение наибольшего из трех чисел

Эти структуры можно комбинировать одну с другой - как путем организации их следований, так и путем создания суперпозиций (вложений одной структуры в другую) - сколь угодно разнообразно для выражения логики алгоритма решения любой задачи. Используя описанные структуры, можно полностью исключить использование каких-либо еще операторов условного и безусловного перехода, что является важным признаком структурного программирования. Направление выполнения команд часто изображают сверху вниз. На рис. 1.24 - 1.26 приведены простейшие примеры структурной реализации алгоритмов работы с величинами.



*Рис. 1.26.* Нахождение суммы 100 чисел

Умение образовывать из базовых структур их суперпозиции в соответствии с условиями конкретной задачи - одно из важнейших в программировании. Допустим, надо ввести в память компьютера 100 чисел и по дороге отсуммировать те из них, которые положительны. Ясно, что ввод - операция циклическая, а внутри этого цикла находится развилка, в которой проверяется знак числа и производится суммирование. Схематически соответствующая суперпозиция изображена на рис. 1.27.

Так как выражение, управляющее циклом, проверяется в самом начале, то в случае, если условие сразу окажется ложным, операторы циклической части «*a*» могут вообще не выполняться. Операторы циклической части «*а*» должны изменять переменную (или переменные), влияющие на значение логического выражения, иначе программа «зациклится» - будет выполняться бесконечно. Рассмотренная циклическая конструкция называется также цикл «пока», или «цикл с предусловием».

Существует и иная конструкция цикла, которая предусматривает проверку условия, по которому, наоборот, выполнение команд циклической части прекращается, после команд циклической части (см. рис. 1.23).



*Рис 1.27* Алгоритм типа развилка, вложенная в цикл,
для нахождения суммы положительных чисел из 100 возможных

Схематические изображения нескольких суперпозиций базовых алгоритмических структур представлены ниже на рис. 1.28-1.31.

Еще одним важным компонентом структурного подхода к разработке алгоритмов является модульность. Модуль - это последовательность логически связанных операций, оформленных как отдельная часть программы. Использование модулей имеет следующие преимущества:

1) возможность создания программы несколькими программистами;

2) простота проектирования и последующих модификаций программы;

3) упрощение отладки программы - поиска и устранения в ней ошибок;

4) возможность использования готовых библиотек наиболее употребительных модулей.

Но, пожалуй, самым важным достижением структурного подхода к разработке алгоритмов является нисходящее проектирование программ, основанное на идее уровней абстракции, которые становятся уровнями модулей в разрабатываемой программе. На этапе проектирования строится схема иерархии, изображающая эти уровни. Схема иерархии позволяет программисту сначала сконцентрировать внимание на определении того, что надо сделать в программе, а лишь затем решать, как это надо делать. При нисходящем проектировании исходная, подлежащая решению задача разбивается на ряд подзадач, подчиненных по своему содержанию главной задаче. Такое разбиение называется детализацией или декомпозицией.

|  |  |
| --- | --- |
|  |  |
| *Рис. 1.28.* Алгоритм типа «цикл, вложенный в неполную развилку» | *Рис. 1.29.* Алгоритм типа «цикл в цикле» |

|  |  |
| --- | --- |
|  |  |
| *Рис. 1.30.* Алгоритм типа «развилка в развилке» | *Рис. 1.31.* Иллюстрация трехкратного вложения одной базовой структуры в другую |

На следующем этапе эти задачи, в свою очередь, разбиваются на более мелкие подчиненные подзадачи и так далее, до уровня относительно небольших подзадач, вторые требуют для решения небольших модулей в 3 - 5 строк. Такой метод роектирования программ позволяет преодолевать проблему сложности разработки программы (и ее последующей отладки и сопровождения).

### 8.3. НОВЕЙШИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММ ДЛЯ ЭВМ

Структурный подход сыграл огромную роль в программировании и вычислительной технике. С его использованием создан большой запас программного обеспечения, решено множество практически значимых задач. Однако, развитие программирования на этом не остановилось. Сегодня дополняющим структурное программирование, создающим основу для разработки современных аудиовизуаль-программных комплексов стало объектное (иногда говорят объектно-ориентированное) программирование, а противостоящим ему при решении определенных классов задач является декларативное программирование, выраженное двумя разными подходами - функциональным и логическим.

Здесь мы ограничимся кратчайшей характеристикой этих направлений, более детально описанных в гл. 3.

Само структурное программирование, наиболее отчетливо выраженное в языке Паскаль (PASCAL), возникло в ходе развития процедурно-ориентированного подхода, заложенного в исторически первом из языков программирования - Фортране (FORTRAN). Во всех языках этого направления разработчик алгоритма (он же, как правило, и программист) описывает, какими действиями следует реализовать процесс. В основе языков этой группы лежат понятия команд (операторов) и данных. Отдельные группы операторов могут объединяться во вспомогательные алгоритмы (процедуры, подпрограммы).

Объект - основное понятие объектного программирования - в первом приближении похож на процедуру. Однако, процедура (подпрограмма) «оживает» лишь внутри той программы, к которой она относится, а объект может вести себя вполне независимо. Он может относиться к иной предметной области нежели основная программа, быть исполненным в ином стиле. Объекты достаточно причудливо связываются друг с другом, могут перенимать свойства друг у друга («наследование»). В объектно-ориентированном подходе исходная задача представляется как совокупность взаимодействующих объектов. Наиболее популярные реализации объектного программирования созданы на основе языков Паскаль, Бейсик (BASIC).

Декларативный подход в разработке компьютерных программ появился в начале 70-х годов. Он не получил столь широкого применения как процедурный, поскольку был 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. Разумеется, для получения результата этот алгоритм все равно нужен, но он должен порождаться автоматически той системой, которая поддерживает декларативно-ориентированный язык программирования. При логическом варианте такого подхода (прежде всего это относится к языку Пролог, PROLOG) задача описывается совокупностью фактов и правил в некотором формальном логическом языке, при функциональном варианте - в виде функциональных соотношений между фактами (язык Лисп, LISP).

Процедурно-ориентированное программирование развивается и в другом направлении - так называемого, параллельного программирования. 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, в которых специалисты видят будущее вычислительной техники.