ТУБ №35:
Структурное программирование

Чем отличается картина великого художника от детского рисунка или простой антихудожественной мазни?

Можно, конечно, порассуждать о соблюдении пропорций, выборе сюжетов и тому подобных вещах, но всегда найдётся такое направление изобразительного искусства, которое не подойдёт под эти критерии. Но кое-что общее выделить всё же можно: произведения искусства обычно выполняются в условиях некоторых ограничений. Произведением искусства рисунок становится тогда, когда укладывается в некоторые рамки: например, стилистические или концептуальные.

Нельзя сказать, что к концу 1960-х годов в программировании царила полна неразбериха: всё-таки в языках программирования уже было из чего выбирать и даже стало понятно, какие решения удачны, а с какими лучше не связываться. И всё же понимание программирования находилось ещё на очень примитивной стадии.

Конец 1960-х годов — время появления ЭВМ третьего поколения, и именно с их появлением связан рост вычислительных мощностей, доступных для использования программам и их разработчикам. Становится возможным решение более сложных задач и даже новых, ранее неподвластных технике, задач, увеличиваются объёмы обрабатываемых данных. Это закономерно отражается на сложности программ. В какой-то момент становится понятно, что старых подходов к разработке программ не хватает. Появляется структурное программирования.

Структурное программирование — это одна из парадигм программирования, т.е. набор концепций, принципов разработки программ. Основная идея структурного программирования заключается в следующем:

Каждый модуль (участок) программы должен проектироваться с единственным входом и единственным выходом

Вся программа при этом представлена вложенными друг в друга подобными модулями (частями), которые ещё называются *функциональными блоками*.

Смысл правила заключается в том, что фрагмент программы, работа которого всегда начинается с одного и того же входящего в его состав оператора и заканчивается аналогичным образом, намного проще тестировать: на единственном входе можно посмотреть (и даже задать свои собственные) значения переменных с исходными данными, на единственном выходе — получить и сравнить с ожидаемым полученный результат. С несколькими входами или выходами приходилось бы не только «ловить» данные в нескольких местах программы, но и проверять, тот ли выход, который ожидается, был использован тестируемым участком программы.

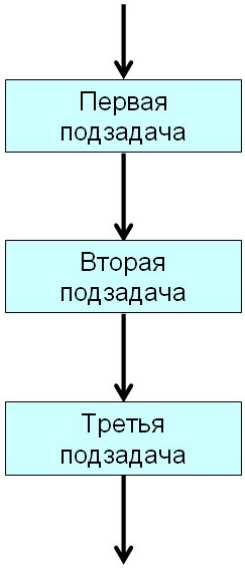
Одной из основ структурного программирования является *принцип Бома-Джакопини*:

Любая программа может быть разработана с использованием трёх базовых структур

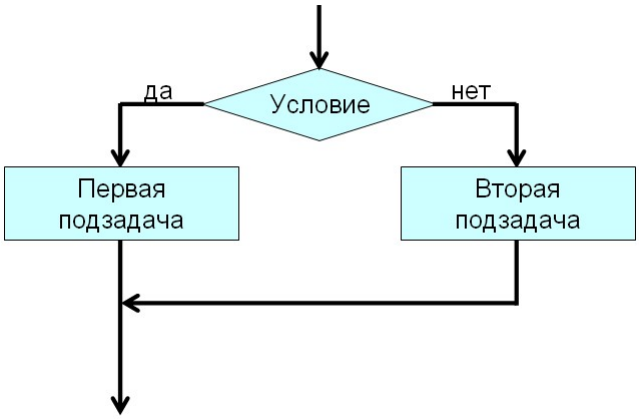
Этими базовыми структурами являются:

- конструкция следования;
- конструкция принятия двоичного решения;
- конструкция обобщённого цикла.

Конструкция следования представляет собой последовательность из одного или нескольких функциональных блоков. На схеме алгоритма ГОСТ 19.701-90 это выглядит примерно так:

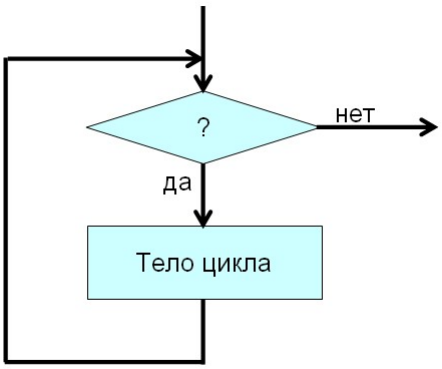


Конструкция принятия двоичного решения позволяет осуществлять выбор одного из двух направлений выполнения алгоритма или программы. Фрагмент схемы алгоритма ГОСТ 19.701-90 для неё выглядит следующим образом:



Следует обратить внимание, что конструкция принятия двоичного решения — это не просто проверка условия с двумя ветвями: важным свойством этой конструкции является то, что по окончании выполнения обеих ветвей выполнение алгоритма или программы продолжается с одного и того же места, что на схеме алгоритма отражено сходящимися в одной точке линиями.

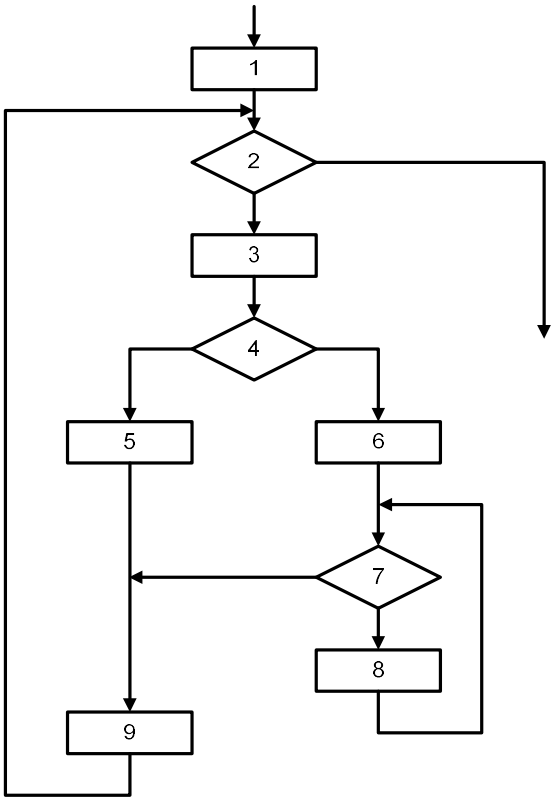
Конструкция обобщённого цикла по сути представляет собой типовой цикл с предусловием:



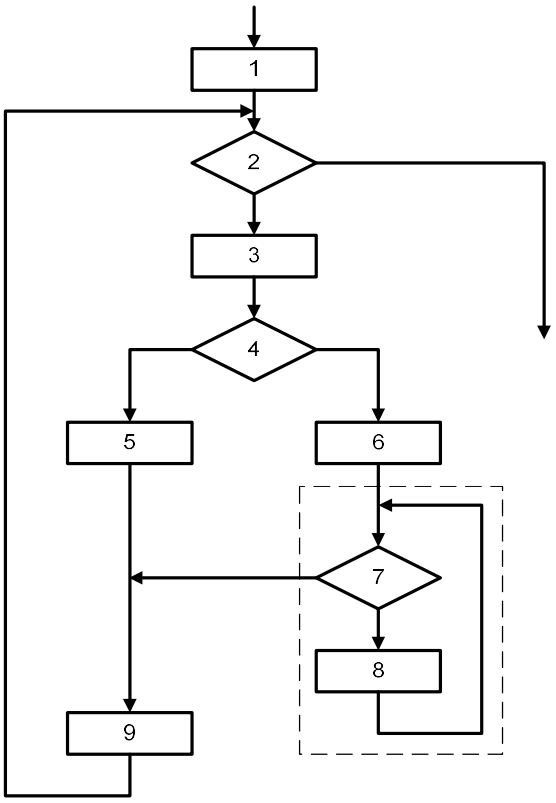
Следует обратить внимание на то, что все три рассмотренные конструкции имеют один вход и один выход. Это позволяет рассматривать их ещё и как функциональные блоки, на чём и основаны преобразования Бома-Джакопини, используемые для того, чтобы проверить (или доказать) *структурированность* алгоритма или программы, т.е. их соответствие принципам структурного программирования, а именно наличие в них на любом уровне детализации исключительно функциональных блоков с одним входом и одним выходом.

Преобразования Бома-Джакопини

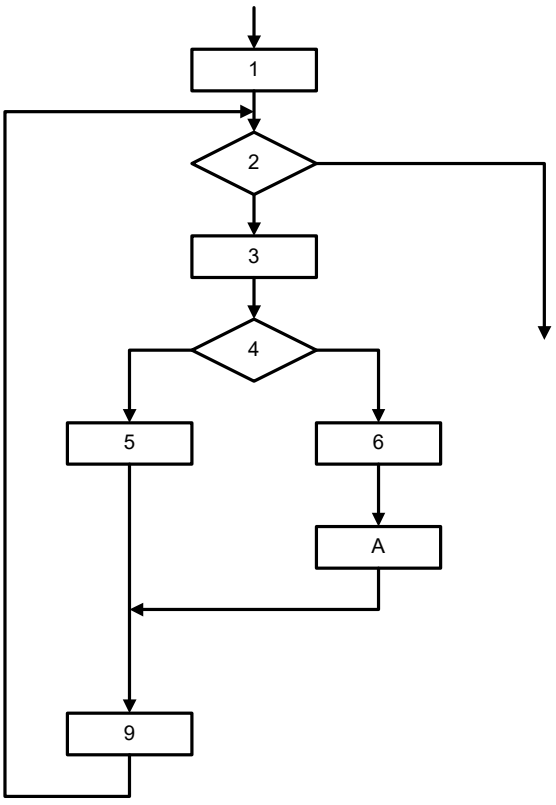
Преобразования Бома-Джакопини проще всего проиллюстрировать на основе схемы алгоритма ГОСТ 19.701-90. Рассмотрим следующий пример алгоритма (выходы символов «Решение» не подписаны, чтобы не загромождать схему):



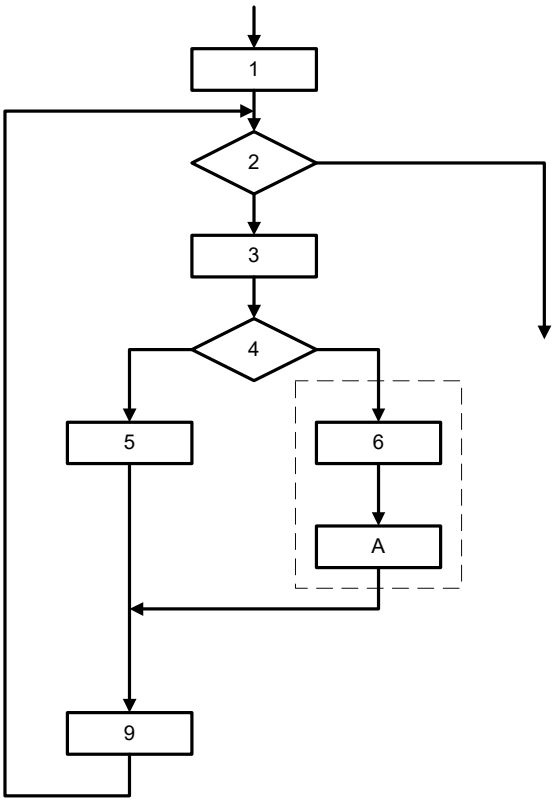
Заметим, что блоки 7 и 8 образуют конструкцию обобщённого цикла: блок 7 является его условием, а блок 8 — телом:



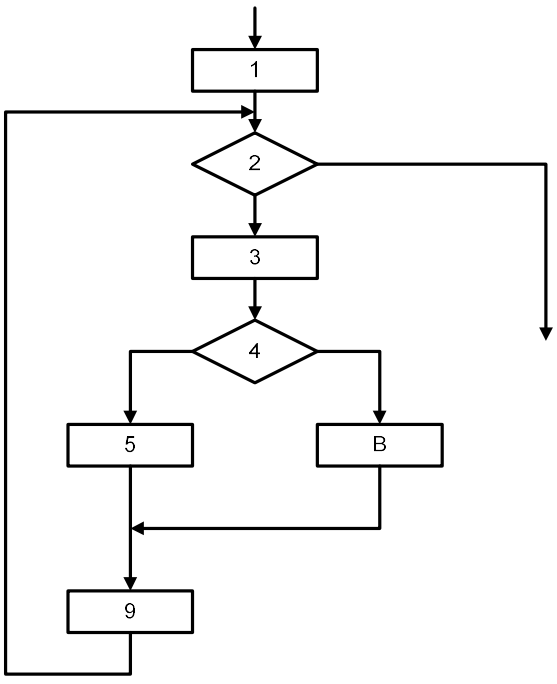
Заменим эту конструкцию на блок А:



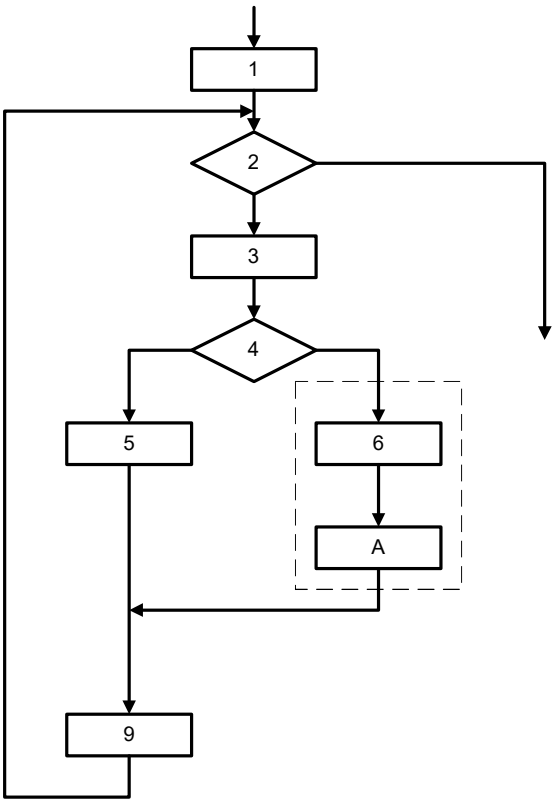
На получившейся в результате такой замены схеме блоки 6 и А образуют конструкцию следования:



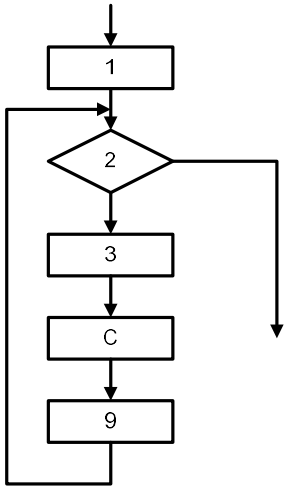
После замены этой конструкции на блок B получится следующая схема:



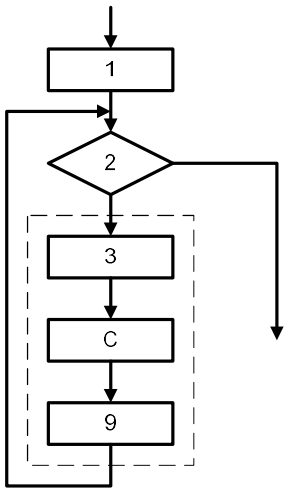
Нетрудно заметить, что в этой схеме блоки 4, 5 и B образуют конструкцию принятия двоичного (иногда ещё говорят «дихотомического») решения:



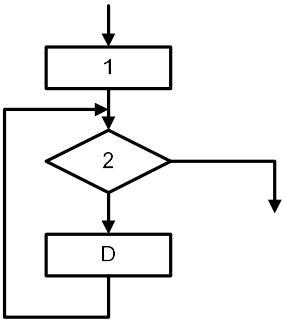
После выполнения соответствующей замены на функциональный блок C получаем схему следующего вида:



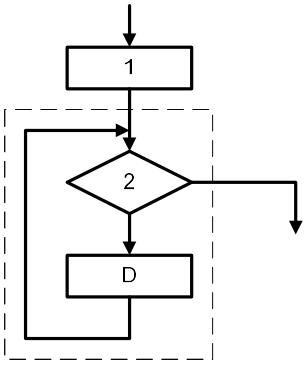
Блоки 3, C и 9 на ней образуют конструкцию следования:



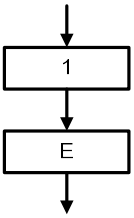
В результате замены этой конструкции на блок D схема приобретает следующий вид:



Теперь блоки 2 и D представляют собой конструкцию обобщённого цикла:



В результате её замены на блок E получим схему, состоящую из двух блоков, образующих конструкцию следования:



После замены этой конструкции на функциональный блок получится схема, состоящая из единственного функционального блока. Это означает, что исходная схема, к которой

применялись преобразования Бома-Джакопини, описывала структурированный алгоритм. Если бы это было не так, на одном из этапов оказалось бы невозможным выделить одну из трёх базовых конструкций.

Преобразования Бома-Джакопини можно также рассматривать и в обратную сторону: начинать с единственного функционального блока, а затем на каждом шаге выполнять замену одного из функциональных блоков на одну из базовых конструкций. Это можно проследить на разобранном только что примере, пройдя его в обратном направлении. Таком подход соответствует нисходящему проектированию и может использоваться для разработки программ, которые будут структурированными «по построению».

Преимущества структурированных программ

Почему нужно стремиться к структурированности программ? Зачем ограничивать себя тремя базовыми конструкциями вместо того, чтобы использовать все доступные возможности? Причин несколько.

Во-первых, как было отмечено ранее, такие программы намного проще тестировать. Любая часть программы может быть протестирована отдельно от других, а затем и в сочетании с другими: благодаря структурированности любая часть такой программы может достаточно легко быть выделена и проанализирована.

Более простое тестирование приводит ко второй причине — повышению производительности труда программистов. Кроме того, тот факт, что программа состоит из ограниченного набора типовых конструкций, позволяет программисту анализировать правильность программы намного быстрее: меньшее количество конструкций легко поддаётся узнаванию и анализу. Всё-таки программа пишется один раз, а читается постоянно.

Третья причина, она же следствие из первых двух: упрощение сопровождения программы. Вносить изменения в программу, которая состоит из простых конструкций, намного проще, чем в программу с запутанной логикой. При этом вместе с изменениями вносится намного меньше ошибок.

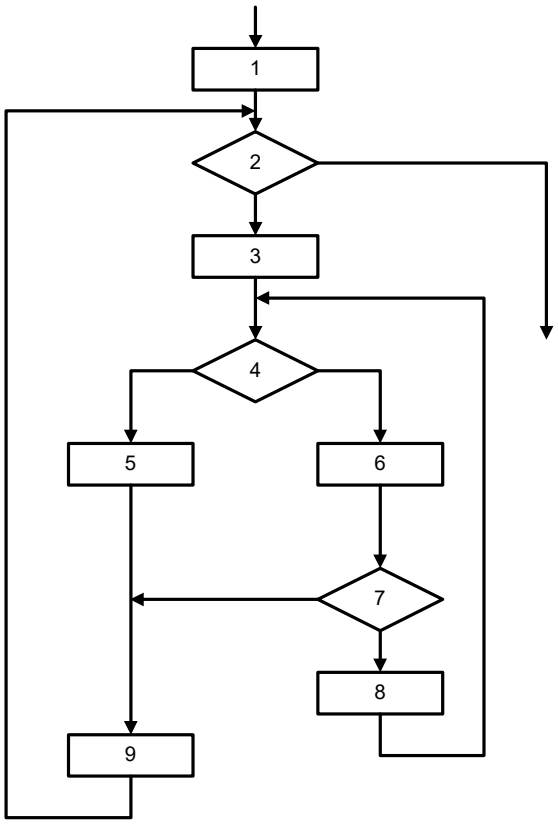
И наконец четвёртая, наиболее ценная для конечного пользователя, причина: генерируемый компилятором машинный код при разработке по принципам структурного программирования оказывается намного эффективнее.

Дело в том, что компилятор, каким бы «умным» он ни казался, по-прежнему остаётся программой, которая кем-то написана и, самое главное, работает по вполне определённому алгоритму. Для того, чтобы генерировать эффективный машинный код, компилятор должен иметь возможность анализировать будущий ход выполнения программы. Безусловно, намного проще это делать для программ, состоящих из типовых конструкций, легко поддающихся анализу. Таким образом, даже более простой компилятор оказывается в состоянии намного лучше оптимизировать именно структурированные программы, не говоря уже о более продвинутых компиляторах.

Хорошая новость заключается в том, что три базовые конструкции Бома-Джакопини в точности соответствуют операторам языков Pascal/Delphi: конструкция принятия двоичного решения — это оператор `if`, а конструкция обобщённого цикла — оператор `while` (и иногда — оператор `for`, который логически также является циклом с предусловием, просто более ограниченным в возможностях ради производительности).

Дополнительные вопросы

1. Является ли следующий алгоритм структурированным? Почему?



2. Докажите, что цикл с постусловием, хоть и не является одной из трёх базовых конструкций по Бому-Джакопини, может быть использован без нарушения принципов структурного программирования.