ТУБ №37:
Метод дублирования кодов

Представьте себе: приходите Вы на работу (программистом, разумеется), открываете исходные коды проекта, который Вам поручено сопровождать, а там…

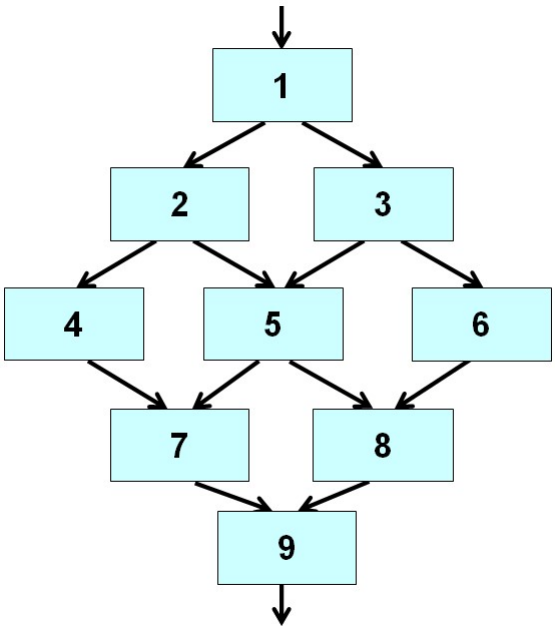
А там, к сожалению, нередко оказывается код, глядя на который хочется рыдать и биться в истерике. Иногда по причине того, что предыдущий разработчик (а попасть на новый проект удаётся нечасто!) работал в условиях очень сжатых сроков, иногда — просто по причине его низкой квалификации. О причинах можно рассуждать долго, а работа ждать не будет: её нужно делать здесь и сейчас.

Один из самых неприятных случаев разгребания авгиевых конюшен — код, нарушающий принципы структурного программирования. Как правило, это случается, когда алгоритм, который пытались записать в коде, изначально довольно запутанный да ещё и менялся несколько раз за последние 10 лет. В этом случае зачастую в спешке доводят код до состояния, когда он начинает выдавать правильные результаты, а приведение его в порядок откладывают до лучших времён. Которые, бывает, не наступают никогда.

Можно, конечно, выбросить такой код и переписать заново, с нуля, но зачастую это не будет оценено начальством, ведь чётко же поставлена задача: «Воплотить в жизнь новые пожелания заказчика». Где там хоть слово про переписывание существующего кода? Он ведь уже есть и в таком виде справляется со своими задачами! И поверьте, переубедить руководство будет очень непросто: для очень редкого начальника качество важнее, чем быстрое завершение проекта. Деньги-то компании где-то нужно брать.

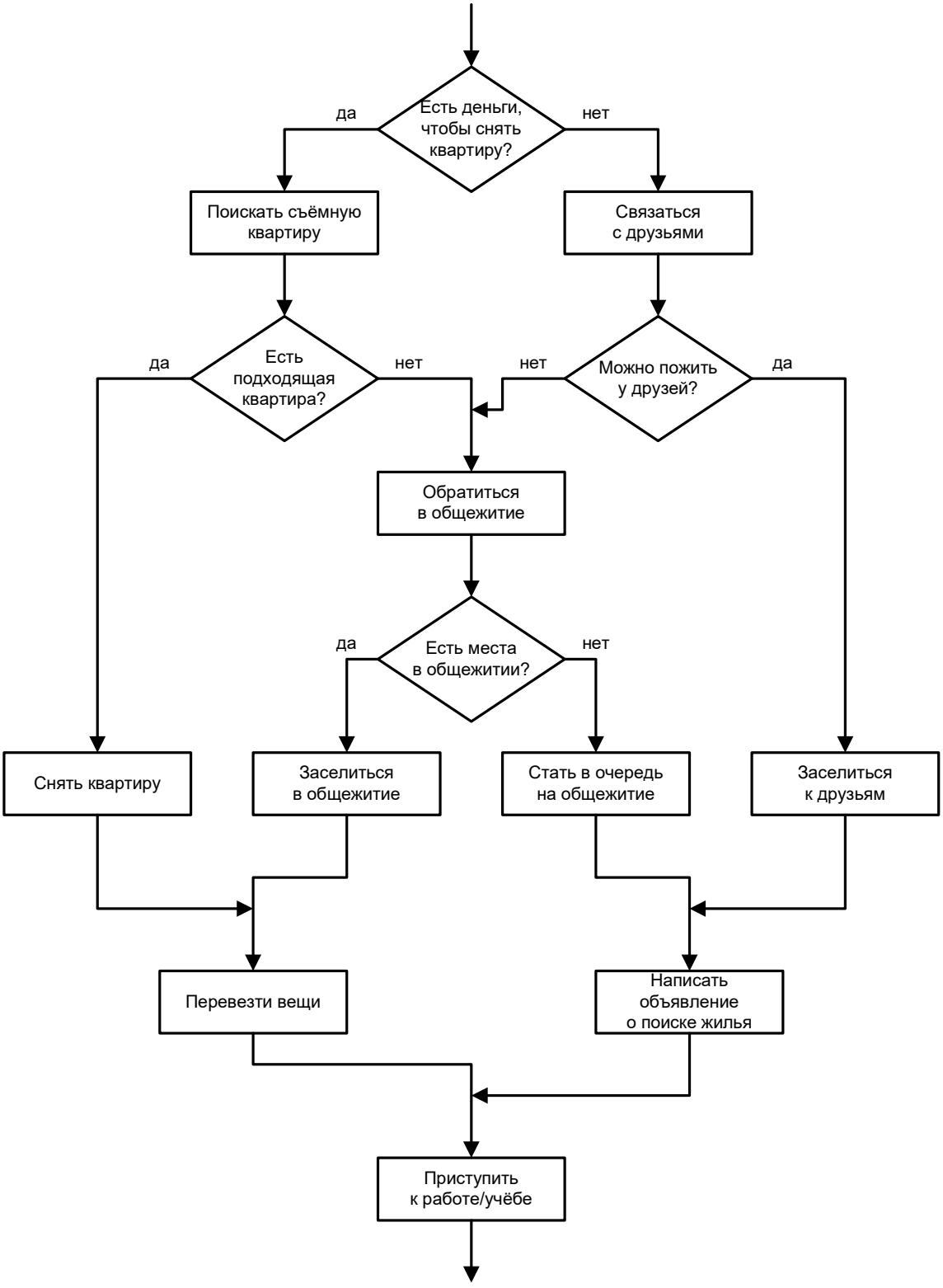
В общем, обычно придётся разбираться с тем, что есть.

Одним из способов «подмести» в коде, не пуская весь «город» под снос, — методов преобразования неструктурированных программ в структурированные — является *метод дублирования кодов*. Чтобы продемонстрировать его суть, представим себе, что у нас есть алгоритм такого вида:



Каждый блоком здесь обозначен какой-то фрагмент алгоритма, стрелки показывают, от каких блоков к каким может передаваться управления.

Критически настроенный читатель, вероятно, заявит, что такие алгоритмы ещё поискать нужно, поэтому приведём пример алгоритма из жизни, который вполне укладывается в эту структуру:



Представьте себе, что Вам предложили работу Вашей мечты (или учёбу в университете Вашей мечты). Отличная новость, не правда ли? Но вот незадача: работать (или учиться) придётся в другом городе. Вполне типовая история для многих тысяч студентов ежегодно.

Конечно, идеальный вариант — найти отдельную квартиру. Жить самостоятельно, без посторонних людей и необходимости подстраиваться под их распорядок дня — мечта многих студентов. Но сразу купить себе квартиру могут лишь единицы, поэтому, как правило, речь идёт о съёмном жилье.

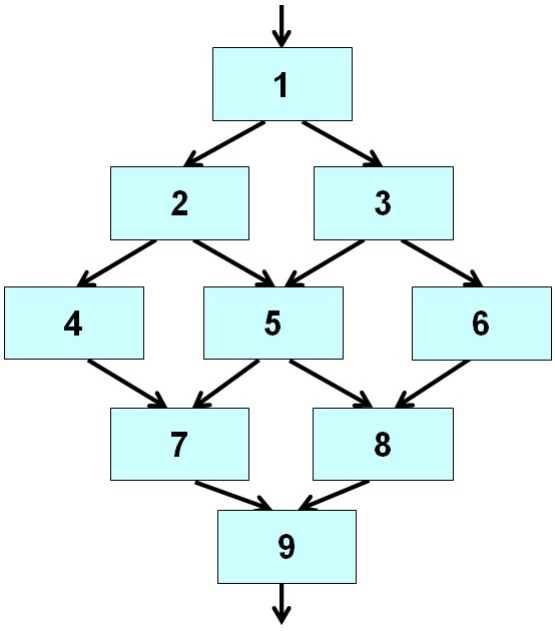
Но снимать квартиру — удовольствие всё же не из дешёвых, поэтому нередко учебные заведения и даже некоторые организации предоставляют своим сотрудникам общежития. Это довольно дешёвый вариант, но жить с посторонними людьми в одном помещении — не так просто, как может показаться. К тому же, иногда в общежитии просто не оказывается свободных мест.

Третий вариант подойдёт везунчикам, у которых в этом городе уже есть друзья, знакомые или родственники, готовые приютить. Это решение обычно всё же временное, но на первых порах и оно может подойти.

Наконец четвёртый вариант — ездить каждый день между городами. В зависимости от расстояния это может быть как лёгкая прогулка, так и утомительное путешествие, и чаще всего именно второе, однако если все остальные варианты не подошли, выбора обычно не остаётся.

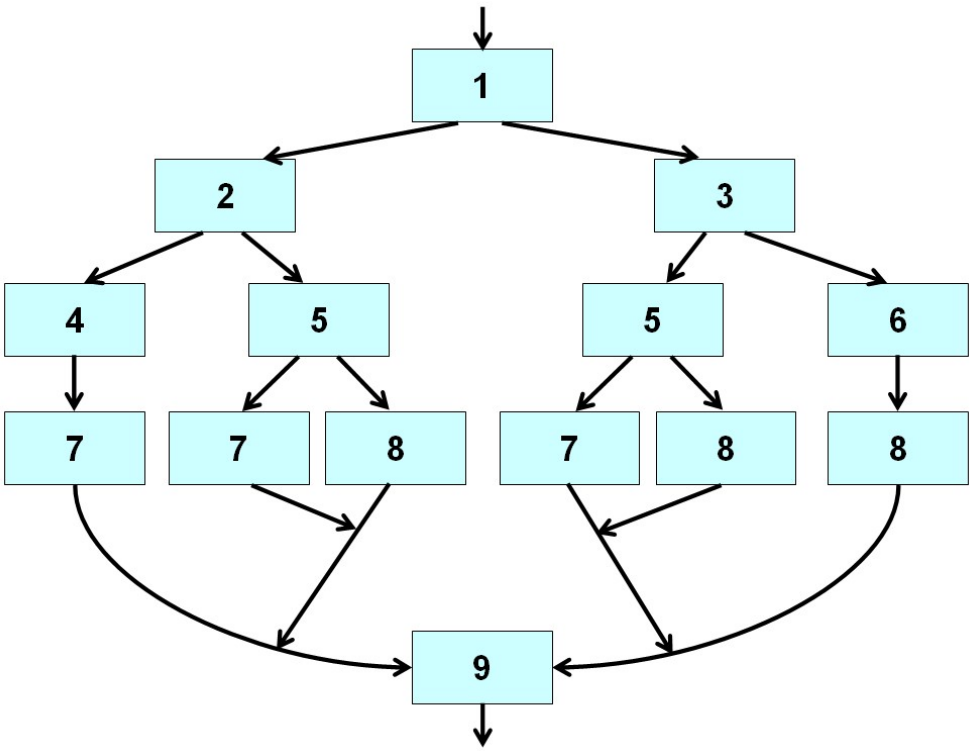
Если посмотреть на схему алгоритма, приведённую чуть выше, можно заметить, что сворачиванию к одному блоку с помощью преобразований Бома-Джакопини она не поддаётся, причём совсем. Следовательно, о структурированности такого алгоритма речи идти не может. Но как его преобразовать в структурированный?

Вернёмся к первоначальной схеме:

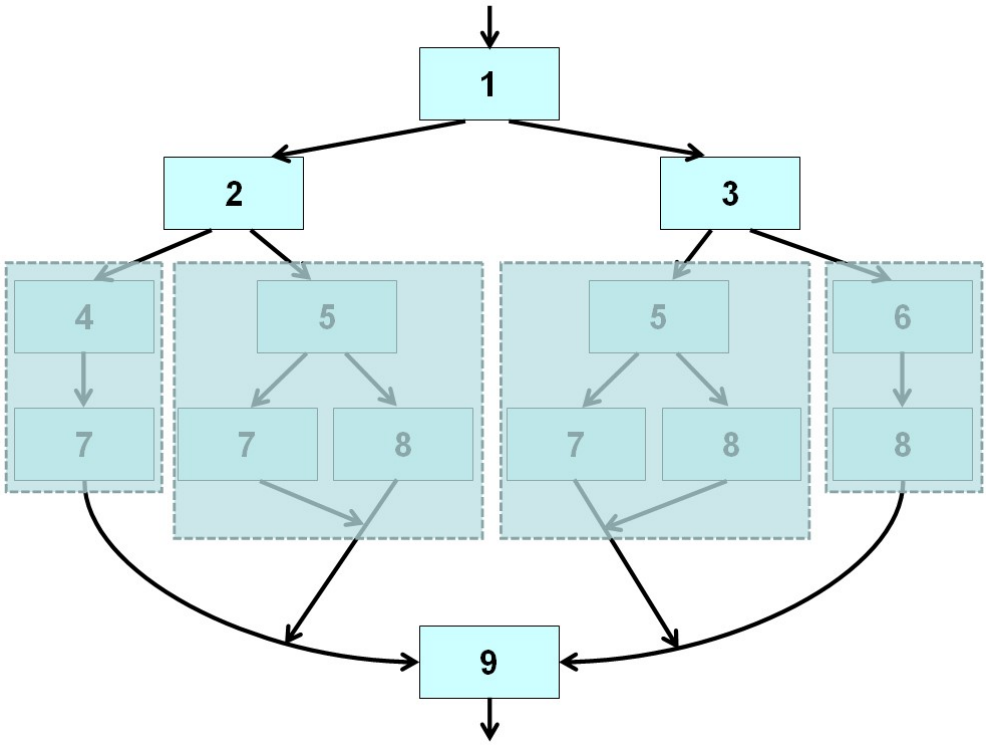


Нетрудно заметить, что она вполне соответствует схеме рассмотренного алгоритма, просто не детализирует конкретных шагов. При этом на ней есть блоки, у которых имеется по два входа. Именно на работе с ними и основан метод дублирования кодов.

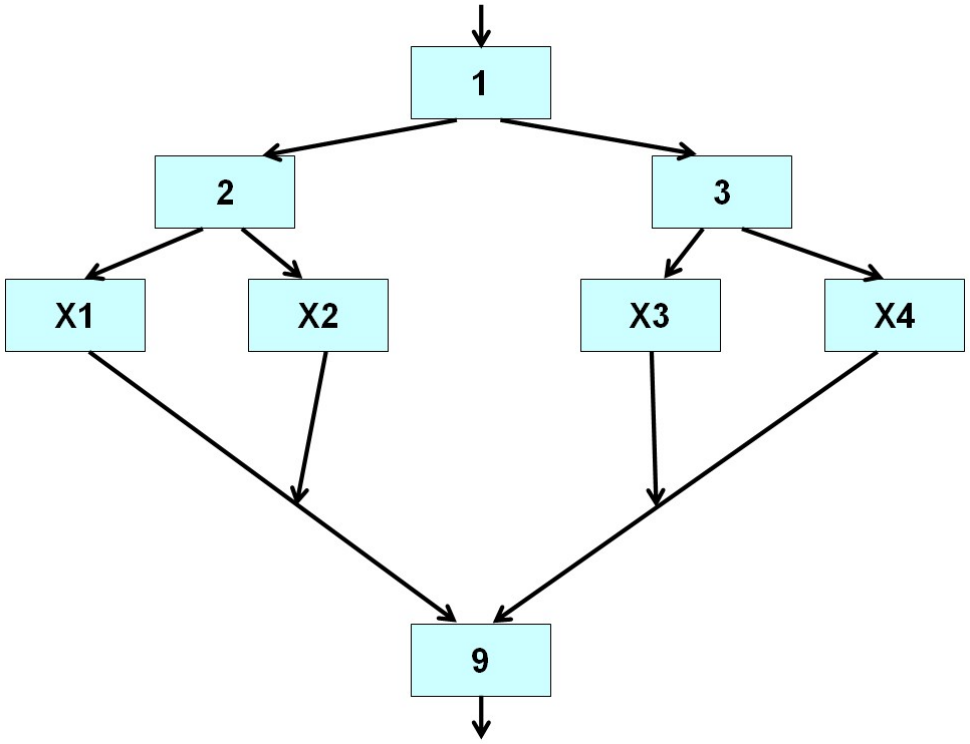
Продублируем все блоки с двумя выходами, кроме самого последнего:



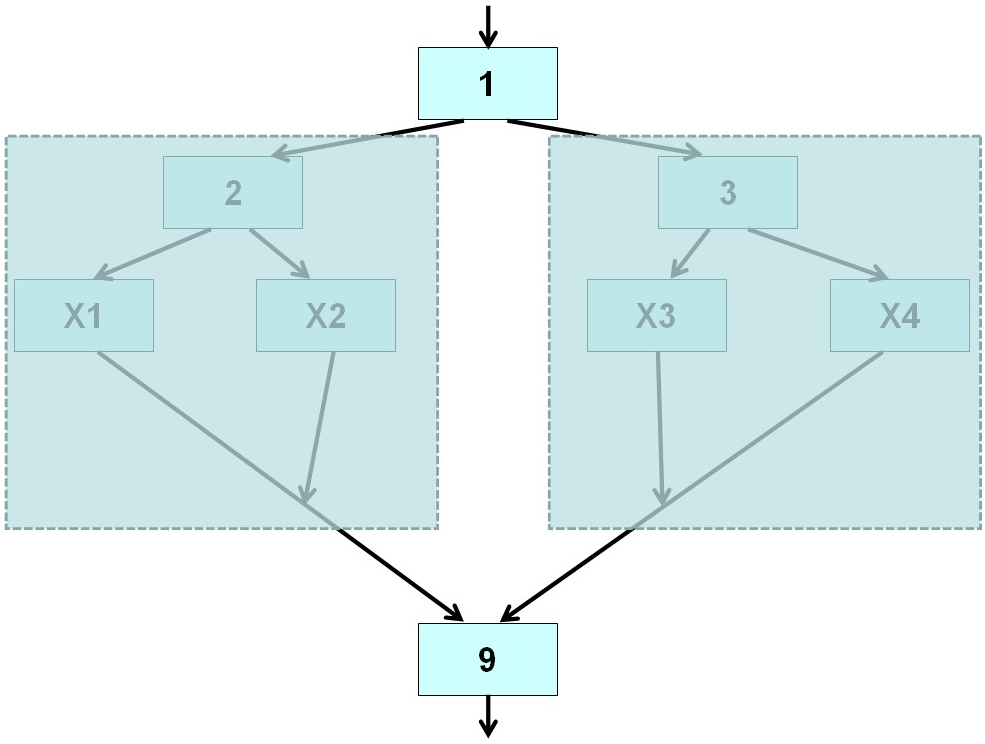
Вспомним, что на схеме алгоритма блокам с двумя выходами (а именно, блокам {2, 3, 5}) соответствовал некоторый набор блоков, включающий блок «Решение», перед которым могли быть некоторые подготовительные действия. Учитывая этот факт, нетрудно заметить, что, например, группы из блоков {5, 7, 8} представляют собой конструкцию принятия двоичного решения, а пары блоков {4, 7} и {6, 8} — конструкции следования:



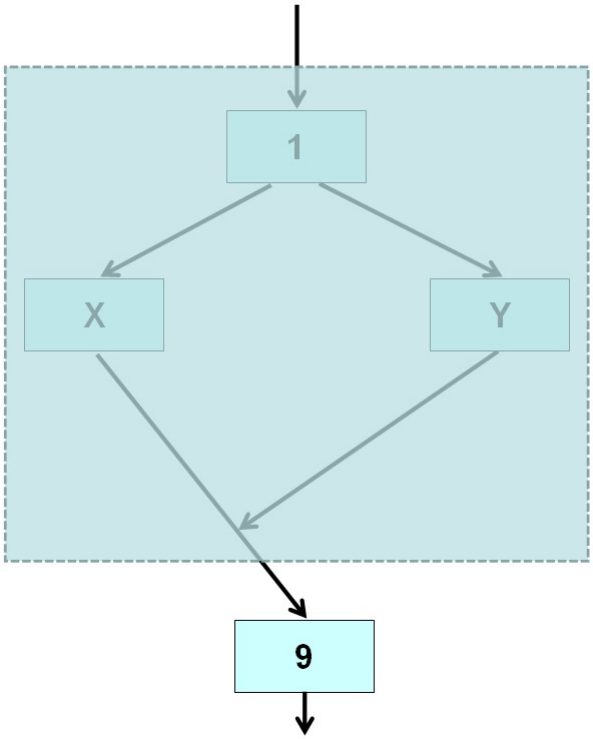
Заменив их функциональными блоками, получим следующую схему:



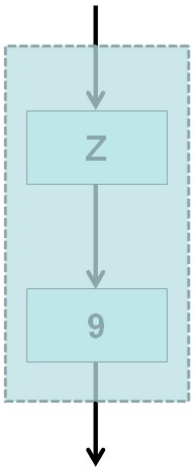
Теперь замечаем, что группы блоков {2, X1, X2} и {3, X3, X4} — точно такие же конструкции, как и те, которые были заменены на предыдущем шаге:



В результате их замены получится ещё более простая схема, на которой также обнаружится подобная конструкция принятия двоичного решения:



После выполнения и этой замены обнаружится, что вся схема свелась к конструкции следования из двух блоков:



После выполнения этой замены будет получен единственный функциональный блок, что доказывает структурированность изначальной схемы — той, которая была получена после дублирования части блоков.

Алгоритмы, подобные изначальному, в коде программы, как правило, строятся с помощью оператора `goto` или аналогичных ему средств языка. Дублированию блоков на схеме при этом соответствует копирование фрагментов кода, принимающих управление, в те места, откуда происходила передача управления.

Основным недостатком данного метода является увеличение объёма кода, причём за счёт повторения его частей: это негативно сказывается на сопровождаемости программы. Ещё одна проблема — его неприменимость к циклическим алгоритмам. Тем не менее, в качестве промежуточного шага для разбора запутанного кода данный метод работает замечательно.

Дополнительные вопросы

1. Преобразуйте следующий фрагмент кода в структурированный с помощью метода дублирования кодов:

```
var
  B1, B2, B3: Boolean;

...

  if B1 then
  begin
    DoA1;
    if B2 then
    begin
      DoA2;
      goto L1;
    end
    else
      goto L2;
  end
  else
  begin
    DoA3;
    if B3 then
    begin
      DoA4;
      goto L3;
    else
      goto L2;
  end;

L2:
  DoA5;
  if B4 then
  begin
    DoA6;
    goto L1;
  end
  else
  begin
    DoA7;
    goto L3;
  end;

L1:
  DoA8;
  goto L4;
L3:
  DoA9;
L4:
  WriteLn('Готово.');
```