ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ

ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ М. Е. ЕВСЕВЬЕВА»

Факультет физико-математический

Кафедра физики, информационных технологий и методик обучения

**РЕФЕРАТ**

СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Автор работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Е. Лемайкина

Направления подготовки 44.03.05 Педагогическое образование

Профиль Физика. Информатика

Руководитель работы

канд. пед. наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т. В. Кормилицына

Оценка \_\_\_\_\_\_\_\_\_\_

Саранск 2022Оглавление

[Введение 3](#_Toc116840632)

[Основные критерии оценки качества программы для ЭВМ. 7](#_Toc116840633)

[Нисходящее проектирование 10](#_Toc116840634)

[Процедуры и функции 11](#_Toc116840635)

[Параметры подпрограмм 11](#_Toc116840636)

[Управление последовательностью вызова подпрограмм 11](#_Toc116840637)

[Структура подпрограммы 12](#_Toc116840638)

[Формальные и фактические параметры: 13](#_Toc116840639)

[Методы и типы структурного программирования 14](#_Toc116840640)

[Структурные операторы 17](#_Toc116840641)

[Заключение 20](#_Toc116840642)

# Введение

Структурное программирование – это технология создания программ, позволяющая путем соблюдения определенных правил уменьшить время разработки и количество ошибок, а также облегчить возможность модификации программы.

Процессор электронно-вычислительной машины умеет выполнять лишь простейшие команды. Каким же образом компьютер решает сложнейшие задачи обработки информации? Для решения этих задач программист должен составить подробное описание последовательности действий, которые необходимо выполнить центральному процессору компьютера. Составление такого пошагового описания процесса решения задачи называется алгоритмизацией, а алгоритмом называется конечный набор правил, расположенных в определённом логическом порядке, позволяющий исполнителю решать любую конкретную задачу из некоторого класса однотипных задач. В разных ситуациях в роли исполнителя может выступать электронное или какое-либо иное устройство или человек.

При создании средних по размеру приложений используется структурное программирование, идея которого заключается в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был ясно виден из исходного текста. Для этого надо иметь сред­ства для создания программы не только с помощью трех простых операторов, но и с помощью средств, более точно отражающих конкретную структуру алгоритма. С этой целью в программирование введено понятие подпрограммы — набора операторов, выполняющих нужное действие и не зависящих от других частей исходного кода. Комбинируя подпрограммы, удается формировать итоговый алгоритм уже не из простых операторов, а из законченных блоков кода, имеющих опреде­ленную смысловую нагрузку, причем обращаться к таким блокам можно по назва­ниям.

История

Методология структурного программирования появилась как следствие возрастания сложности решаемых на компьютерах задач и соответственного усложнения программного обеспечения. В 70-е годы XX века объёмы и сложность программ достигли такого уровня, что "интуитивная" разработка программ, которая была нормой в более раннее время, перестала удовлетворять потребностям практики. Программы становились слишком сложными, чтобы их можно было нормально сопровождать, поэтому потребовалась какая-то систематизация процесса разработки и структуры программ. Наиболее сильной критике со стороны разработчиков структурного подхода к программированию подвергся оператор GOTO (оператор безусловного перехода), имеющийся почти во всех языках программирования. Использование произвольных переходов в тексте программы приводит к получению запутанных, плохо структурированных программ, по тексту которых практически невозможно понять порядок исполнения и взаимозависимость фрагментов.

Следование принципам структурного программирования сделало тексты программ, даже довольно крупных, нормально читаемыми. Серьёзно облегчилось понимание программ, появилась возможность разработки программ в нормальном промышленном режиме, когда программу может без особых затруднений понять не только её автор, но и другие программисты. Это позволило разрабатывать достаточно крупные для того времени программные комплексы силами коллективов разработчиков, и сопровождать эти комплексы в течение многих лет, даже в условиях неизбежной ротации кадров.

Методология структурной разработки программного обеспечения была признана "самой сильной формализацией 70-х годов". После этого слово "структурный" стало модным в отрасли, его начали использовать везде, где надо и не надо. Появились работы по "структурному проектированию", "структурному тестированию", "структурному дизайну" и так далее, в общем, произошло примерно то же самое, что происходило в 90-х годах и происходит в настоящее время с терминами "объектный" и "объектно-ориентированный".

В настоящее время, когда сфера применения ЭВМ чрезвычайно расширилась, разработка и эксплуатация программ осуществляется, как правило, разными людьми. Поэтому наряду с эффективностью на первый план выдвигаются и другие важные характеристики программ такие, как понятность, хорошая документированность, надежность, гибкость, удобство сопровождения и т.п.

Проблема разработки программ, обладающих такими качествами, объясняется трудоемкостью процесса программирования и связанным с этим быстрым ростом стоимости программного обеспечения.

Для создания "хорошей" программы появляется необходимость придерживаться определенных принципов или определенной дисциплины программирования. Значительный прогресс в области программирования достигается с использованием так называемого структурного программирования.

Появление новой технологии, или, как еще говорят, дисциплины программирования, основанной на структурном подходе, связано с именем известного голландского ученого Э.Дейкстры (1965 г.). В своих работах он высказал предположение, что оператор GOTO может быть исключен из языков программирования и что квалификация программиста обратно пропорциональна числу операторов GOTO в его программах. Такая дисциплина программирования упрощает и структурирует программу.

Однако представление о структурном программировании, как о программировании без использования оператора GOTO, является ошибочным. Например, Хоор определяет структурное программирование как "систематическое использование абстракции для управления массой деталей и способ документирования, который помогает проектировать программу".

Структурное программирование можно толковать как "проектирование, написание и тестирование программы в соответствии с заранее определенной дисциплиной".

Структурный подход к программированию как раз и имеет целью снижение трудоемкости всего процесса создания программного обеспечения от технического задания на разработку до завершения эксплуатации. Он означает необходимость единой дисциплины на всех стадиях разработки программы. В понятие структурного подхода к программированию обычно включают нисходящие методы разработки программ (принцип «сверху вниз»), собственно структурное программирование и так называемый сквозной структурный контроль.

Основной целью структурного программирования является уменьшение трудностей тестирования и доказательства правильности программы. Это особенно важно при разработке больших программных систем. Опыт применения методов структурного программирования при разработке ряда сложных операционных систем показывает, что правильность логической структуры системы поддается доказательству, а сама программа допускает достаточно полное тестирование. В результате в готовой программе встречаются только тривиальные ошибки кодирования, которые легко исправляются.

Структурное программирование улучшает ясность и читабельность программ. Программы, которые написаны с использованием традиционных методов, особенно те, которые перегружены операторами GOTO, имеют хаотичную структуру.

Структурированные программы имеют последовательную организацию, поэтому возможно читать такую программу сверху донизу без перерыва. Наконец, структурное программирование призвано улучшить эффективность программ.

# Основные критерии оценки качества программы для ЭВМ.

Известно, что один и тот же алгоритм может быть реализован на ЭВМ различными способами, т.е. может быть составлено несколько различных программ, решающих одну и ту же задачу.

Таким образом, нужно иметь некоторые критерии оценки программы, с помощью которых можно судить насколько одна программа лучше другой. Анализ и оценка программы носят преимущественно качественный характер.

1. Программа работает и решает поставленную задачу. Понятно, что эта характеристика программы является самой важной.

В связи с этим каждая программа должна быть устроена так, чтобы можно было проверить правильность полученных результатов. Такая проверка проводится в процессе отладки программы, на определенных наборах входных данных, для которых заранее известен ответ. Но отладка может доказать лишь наличие ошибок в программе, но не может доказать правильности программы для всех возможных вычислений, реализуемых с ее помощью. В связи с этим необходима разработка методов аналитической верификации программы.

Для аналитического доказательства правильности программы требуется, чтобы программа легко анализировалась. Это означает, что программа должна быть устроена так, чтобы можно было понять, каким образом с ее помощью получается данный ответ.

2. Минимальное время, затрачиваемое на тестирование и отладку программы. Тестирование и отладка программы – необходимый этап в процессе решения задачи на ЭВМ. Он занимает от трети до половины всего времени разработки программы, поэтому очень важно уменьшить время, затрачиваемое на тестирование и отладку.

Тестирование и отладка программы облегчается, если программа просто анализируется и снабжена необходимыми комментариями, облегчающими ее понимание. Хорошие комментарии могут ускорить процесс отладки.

Понимание и отладка программы облегчается, если она имеет простую и ясную структуру, в частности, если ограничено использование операторов передачи управления (GOTO). Перегруженность программы этими операторами приводит к хаотической структуре и затрудняет отладку.

Еще один важный принцип – использование мнемонических обозначений для переменных. Языки программирования представляют здесь вполне достаточные возможности. Для лучшего понимания программы необходимо использовать мнемонику, отражающую физический (математический, экономический и т.д.) смысл переменной (например, SPEED - скорость).

3. Уменьшение затрат на сопровождение. Разработанная и отлаженная программа предназначена для многократного использования, и ее эксплуатацией, как правило, занимаются не разработчики, а другие программисты, входящие в так называемую группу сопровождения.

Программистам, сопровождающим программу, часто приходится продолжать отладку программы и производить ее модернизацию, в связи с изменением технического задания, введением новых средств программного обеспечения или выявлением новых ошибок и недоработок в программе.

Для уменьшения затрат на сопровождение необходимо, чтобы каждый разработчик учитывал сложность сопровождения. Следует разрабатывать, отлаживать и оформлять программу с учетом того, что ее будут использовать и сопровождать другие программисты.

4. Гибкость программы. Разработанная программа обычно находится в эксплуатации длительное время. За это время могут измениться требования к решаемой задаче, техническое задание, требования к программе. Появляется необходимость внести определенные изменения в программу, что в некоторых случаях бывает трудно сделать, т.к. разработчиком не предусмотрена такая возможность. "Хорошая" программа должна допускать модификацию.

5. Уменьшение затрат на разработку. Программирование является коллективным трудом. Состав группы программистов, работающих над решением данной задачи, может по каким-либо причинам измениться. Поэтому проектирование и разработка программы должны вестись таким образом, чтобы было возможно при необходимости передать ее завершение другому программисту. Несоблюдение этого требования часто приводит к срыву сроков сдачи программ в эксплуатацию.

6. Простота и эффективность. Программа должна быть просто организована.

Это может проявляться и в структуре программы, и в использовании простых и наиболее естественных средств языка программирования, и в предпочтении простых структур данных и т.п.

Эффективность программы считается одной из главных ее характеристик.

Поэтому часто в ущерб другим качествам программы разработчики прибегают к сложным ухищрениям, чтобы уменьшить объем используемой памяти или сократить время выполнения программы. Во многих случаях затрачиваемые на это усилия не оправдывают себя. Разумный подход к повышению эффективности программы состоит в том, чтобы выявить наиболее "узкие" места и постараться их улучшить.

# Нисходящее проектирование

Наличие подпрограмм позволяет вести проектирование и разработку приложения сверху вниз — такой подход называется нисходящим проектированием. Сначала выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть и завершение), потом каждый из этих модулей детализируется на более низком уровне, разбиваясь в свою очередь на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется реализованной.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность некоторые подпрограммы не реализовывать сразу, а временно откладывать, пока не будут закончены другие части. Например, если имеется необходимость вычисления сложной математической функции, то выделяется отдельная подпрограмма такого вычисления, но реализуется она временно одним оператором, который просто присваивает заранее выбранное значение. Когда все приложение будет написано и отлажено, тогда можно приступить к реализации этой функции.

Немаловажно, что небольшие подпрограммы значительно проще отлаживать, что существенно повышает общую надежность всей программы.

Очень важная характеристика подпрограмм — это возможность их повторного использования. С интегрированными системами программирования поставляются большие библиотеки стандартных подпрограмм, которые позволяют значительно повысить производительность труда за счет использования чужой работы по созда­нию часто применяемых подпрограмм.

# Процедуры и функции

Подпрограммы бывают двух видов — процедуры и функции. Отличаются они тем, что процедура просто выполняет группу операторов, а функция вдобавок вычисляет некоторое значение и передает его обратно в главную программу (возвращает значение). Это значение имеет определенный тип (говорят, что функция имеет такой-то тип).

В Си++ понятия «процедура» нет — там имеются только функции, а если никакого значения функция не вычисляет, то считается, что она возвращает значение типа «никакое» (void).

Параметры подпрограмм

Чтобы работа подпрограммы имела смысл, ей надо получить данные из внешней программы, которая эту подпрограмму вызывает. Данные передаются подпрограмме в виде параметров или аргументов, которые обычно описываются в ее заголовке ' так же, как переменные.

Управление последовательностью вызова подпрограмм

Подпрограммы вызываются, как правило, путем простой записи их названия с нужными параметрами. В Бейсике есть оператор CALL для явного указания того, что происходит вызов подпрограммы.

Подпрограммы активизируются только в момент их вызова. Операторы, находящиеся внутри подпрограммы, выполняются, только если эта подпрограмма явно вызвана. Пока выполнение подпрограммы полностью не закончится, оператор главной программы, следующий за командой вызова подпрограммы, выполняться не будет.

Подпрограммы могут быть вложенными — допускается вызов подпрограммы не только из главной программы, но и из любых других подпрограмм.

В некоторых языках программирования допускается вызов подпрограммы из себя самой. Такой прием называется рекурсией и потенциально опасен тем, что может привести к зацикливанию — бесконечному самовызову.

Структура подпрограммы

Подпрограмма состоит из нескольких частей: заголовка с параметрами, тела подпрограммы (операторов, которые будут выполняться при ее вызове) и завершения подпрограммы.

Локальные переменные, объявленные внутри подпрограммы, имеют областью действия только ее тела.

Функции:

|  |  |  |
| --- | --- | --- |
|  | Бейсик | Паскаль |
| Заголовок функции | FUNCTION имя (список параметров)  Тип возвращаемого значения определяется специальным символом после имени функции | function имя тип\_функции  функции |
| Тело | Последовательность операторов | Begin  Последовательность операторов  end |
| Завершение | END FUNCTION | нет |
| Завершение | END FUNCTION | нет |

Процедуры

|  |  |  |
| --- | --- | --- |
|  | Бейсик | Паскаль |
| Заголовок процедуры | Sub имя | Procedure имя |
| Тело | Последовательность | Begin  Последовательность операторов  end; |
| Завершение | END SUB |  |

# Формальные и фактические параметры:

Во время создания подпрограммы заранее не известно, какие конкретно параметры она может и будет получать. Поэтому в качестве переменных, выступающих в роли ее аргументов в заголовке, могут использоваться произвольные допустимые названия, даже совпадающие с уже имеющимися. Компилятор все равно поймет, что это не одно и то же.

Параметры, которые указываются в заголовке подпрограммы, называются формальными. Они нужны только для описания тела подпрограммы. А параметры (конкретные значения), которые указываются в момент вызова подпрограммы, назы­ваются фактическими параметрами. При выполнении операторов подпрограммы формальные параметры как бы временно заменятся на фактические.

Пример.

int а, у;

а = 5;

у = SQR(a);

Программа вызывает функцию SQR() с одним фактическим параметром а. Внутри подпрограммы формальный параметр х получает значение переменной а и возводится в квадрат. Результат возвращается обратно в программу и присваивается переменной у.

# Методы и типы структурного программирования

К основным методам структурного программирования относится, прежде всего, отказ от бессистемного употребления оператора GOTO и преимущественное использование других структурированных операторов, методы нисходящего проектирования разработки программы, идеи пошаговой детализации и некоторые другие соглашения, касающиеся дисциплины программирования.

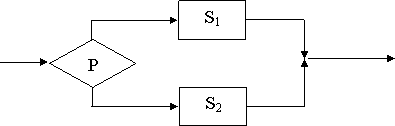
Всякая программа, в соответствии с структурным подходом к программированию, может быть построена только с использованием трех основных типов блоков.



1. Функциональный блок, который на блок-схеме изображается в виде прямоугольников с одним входом и одним выходом:

Функциональному блоку в языках программирования соответствуют операторы ввода и вывода или любой оператор присваивания.

В виде функционального блока может быть изображена любая последовательность операторов, выполняющихся один за другим, имеющая один вход и один выход.



2. Условная конструкция. Этот блок включает проверку некоторого логического условия (P), в зависимости от которого выполняется либо один

(S1), либо другой (S2) операторы:

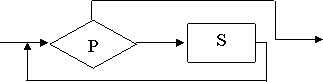
На языке "Е-практикума":

. если

. . то

. . иначе

. все



3. Блок обобщенного цикла. Этот блок обеспечивает многократное повторение выполнения оператора S пока выполнено логическое условие P:

На языке "Е-практикума" циклы реализуются 2 способами. Цикл с параметром:

. нц для от до

. .

. . ...

. кц

Цикл с предусловием:

. нц пока

. .

. . ...

. кц

При конструировании программы с использованием рассмотренных типов блоков эти блоки образуют линейную цепочку так, что выход одного блока подсоединяется ко входу следующего. Таким образом, программа имеет линейную структуру, причем порядок следования блоков соответствует порядку, в котором они выполняются.

Такая структура значительно облегчает чтение и понимание программы, а также упрощает доказательство ее правильности. Так как линейная цепочка блоков может быть сведена к одному блоку, то любая программа может, в конечном итоге, рассматриваться как единый функциональный блок с один входом и одним выходом.

При проектировании и написании программы нужно выполнить обратное преобразование, то есть этот блок разбить на последовательность подблоков, затем каждый подблок разбить на последовательность более мелких блоков до тех пор, пока не будут получены "атомарные" блоки, рассмотренных выше типов. Такой метод конструирования программы принято называть нисходящим

("сверху вниз").

При нисходящем методе конструирования алгоритма и программы первоначально рассматривается вся задача в целом. На каждом последующем этапе задача разбивается на более мелкие подзадачи, каждая подзадача, в конечном итоге на еще более мелкие подзадачи и так до тех пор, пока не будут получены такие подзадачи, которые легко кодируются на выбранном языке программирования. При этом на каждом шаге уточняются все новые и новые детали ("пошаговая детализация").

В процессе нисходящего проектирования сохраняется строгая дисциплина программирования, то есть разбиение на подзадачи осуществляется путем применения только рассмотренных типов конструкций (функциональный блок, условная конструкция, обобщенный цикл), поэтому, в конечном итоге, получается хорошо структурированная программа.

На языке "Е-практикума" последовательную детализацию можно реализовать в виде вспомогательного алгоритма (подпрограммы, процедуры, функции).

... нач

. вспомогательный\_алгоритм(...)

. ... кон алг [] вспомогательный\_алгоритм(...) дано ... надо нач

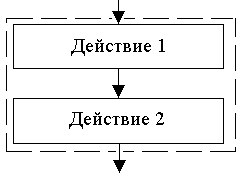
. ... кон

В настоящее время в помощь структурному подходу к прораммированию появилось новое направление - объектное программирование, не отменяющее, а дополняющее принципы структурного подхода, позволяющее разрабатывать и модернизировать программный код с меньшими затратами времени, распределять задачу программирования между большим количеством программистов.

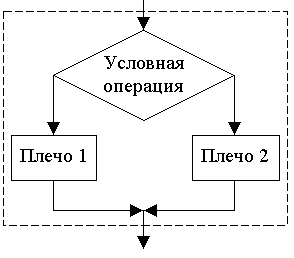
# Структурные операторы

Основная идея структурного программирования заключаются в том, что существует только четыре структурных оператора. Используя эти структурные операторы, можно построить сколь угодно сложную программу.

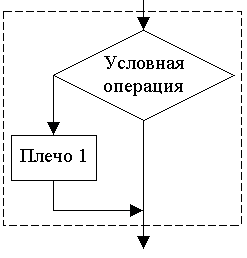
Первый структурный оператор называется линейная цепочка операторов. Любая задача может быть разбита на несколько подзадач. Выполнение подзадач может быть поручено подпрограмме, в названии которой можно (и нужно) отразить подзадачу, которую должна решать эта подпрограмма. На момент написания алгоритма (и программы) верхнего уровня нас не интересует, поэтому вместо настоящей подпрограммы поставим подпрограмму-заглушку.



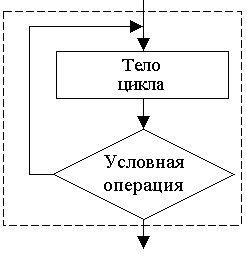
Второй структурный оператор называется условный оператор. Достаточно часто одна или другая задачи должны исполняться в зависимости от определённого условия, которое зависит от результатов выполнения предыдущей программы или от внешних устройств. Каждая из таких задач называется плечом условного оператора.



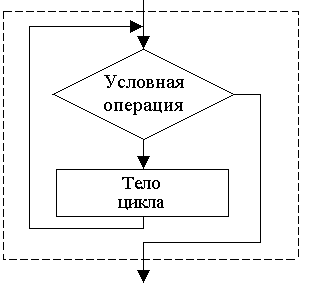
Условный оператор может использоваться в неполном варианте, когда одно из плеч алгоритма отсутствует:



Третий структурный оператор - это оператор цикла с проверкой условия после тела цикла. Такой оператор легко реализуется на языке программирования ассемблер при помощи команды условного или безусловного перехода. Отличие от условного оператора заключается в том, что передача управления осуществляется не вперёд, а назад. На языках программирования высокого уровня такой оператор входит в состав языка (оператор do..while в языке программирования C или оператор repeat..until в языке программирования PASCAL).



Четвёртый структурный оператор - это оператор цикла с проверкой условия до тела цикла. В отличие от предыдущего оператора тело цикла в этом операторе может ни разу не выполниться, если условие цикла сразу же выполнено. Этот оператор, как и условный оператор, невозможно реализовать на одной машинной команде.



|  |  |
| --- | --- |
| Типы управляющей структуры | Применение управляющей структуры |
| Последовательность  Действие 1  Действие 2  Конец | Последовательность включает фиксированный перечень действий (операторов). Каждое очередное действие обрабатывается после завершения предыдущего без дополнительных условий.  Для изменения порядка обработки блоков редактируется последовательность выполняемых |
| Альтернатива (условие выбора)  Начало  Да Условие Нет  Альтернатива1 Альтернатива2  Конец | В блоке Условие содержится условие выбора альтернативы обработки. Каждая альтернатива выполняется 1 раз; выполнение одной из двух альтернатив - обязательно.  Развитие данного типа структуры является множественная альтернатива, когда последовательно проверяются условия выполнения определенных альтернатив. Если очередное условие истинно, обрабатывается соответствующая ему альтернатива, после чего происходит выход. В противном случае - переход к проверке условия следующей альтернативы.  Если ни одно из условий не выполнилось, происходит выход. |
| Цикл ("пока")  Начало  Условие  Нет Да  Тело цикла  Конец | В блоке Условие задается условие тела цикла - определенной обработки. Если условие не выполняется, цикл прерывается и осуществляется выход.  Условие может содержать счетчик повторений тела цикла либо логическое условие.  Тело цикла - произвольная последовательность блоков (операторов) обработки |

# Заключение

Современный этап развития общества характеризуется внедрением информационных технологий во все сферы человеческой деятельности. Новые информационные технологии оказывают существенное влияние и на сферу образования. Происходящие фундаментальные изменения в системе образования вызваны новым пониманием целей, образовательных ценностей, а также необходимостью перехода к непрерывному образованию, разработкой и использованием новых технологий обучения, связанных с оптимальным построением и реализацией учебного процесса с учетом гарантированного достижения дидактических целей. 

Список использованной литературы:

1. [https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5](https://ru.wikipedia.org/wiki/Структурное_программирование).
2. <https://studopedia.ru/9_6278_strukturnoe-programmirovanie.html>
3. https://intuit.ru/studies/courses/40/40/lecture/1208