МИНОБРНАУКИ РФ

ФГБОУ ВО Тверской государственный технический университет

Кафедра “Программное обеспечение”

Курсовая работа

дисциплина «Теоретическая информатика»

Тема: Правила качественного программирования

Выполнили: студенты группы

«Б.ПИН.РИС-21.06»

Латыпов Тимофей

Симонов Александр

Громова Полина

Пономарев Вадим

Черепан Матвей

Проверил: Биллиг Владимир Арнольдович

Тверь,2022

**Оглавление**

1. **Введение 3**
2. **Правила качественного программирования**
   1. **Декомпозиция 3**
   2. **Модульность 3**
   3. **Корректность 3**
   4. **Тестируемость 4**
   5. **Устойчивость 4**
   6. **Стилизованность 5**
   7. **Читаемость 5**
   8. **Универсальность 5**
3. **Практика качественного программирования**
   1. **Декомпозиция 6**
   2. **Модульность 7**
   3. **Корректность 12**
   4. **Тестируемость 12**
   5. **Устойчивость 13**
   6. **Стилизованность 13**
   7. **Читаемость 13**
   8. **Универсальность 13**
4. **Вывод 13**

# **1.Введение**

Правила качественного программирования – фундаментальные принципы программирования, определяющие то, как и зачем нужно проектировать структуру программы, алгоритмов, а также общие подходы к разработке.

# **2.Правила качественного программирования**

**2.1. Декомпозиция**

Декомпозиция – мыслительная операция по разделению целого на части. Это общий приём, применяемый при решении различных проблем, состоящий в разделении задачи на множество частных, тривиальных задач, не превосходящих по сложности исходную проблему, с помощью объединения решений которых, можно сформировать решение исходной проблемы в целом.

Программисты часто сталкиваются с очень абстрактными и сложными задачами, которые предполагают работу в команде. При командном подходе к разработке ПО, использование декомпозиции позволяет давать разработчикам конкретные небольшие задачи и постепенно выстраивать общую структуру приложения из небольших частиц. Также это увеличивает продуктивность разработки, так как появляется возможность параллельно решать несколько задач, и совмещать их на дальнейших этапах.

**2.2. Модульность**

Модульность – приём в программировании, заключающийся в разбиении программы на функционально законченные фрагменты, оформляемые в виде отдельного файла или поименованной непрерывной части программы. При таком подходе программа организована как совокупность небольших независимых блоков, структура и поведение которых подчиняется определённым правилам. Причём каждый модуль выполняет одну определённую задачу.

При написании объёмных комплексных решений, программисты сталкиваются с проблемами тестирования и обнаружения ошибок. Модульность как бы проливает свет на внутреннюю работу алгоритма, расставляет всё по своим местам и противостоит неявной логике и запутанности. Она позволяет быстро обнаружить модуль, содержащий ошибку и исправить её.

Модульность так же позволяет проще понимать общую логику решения и то, каким образом достигается результат.

**2.3. Корректность**

Под корректностью программы понимается её соответствие некоторому эталону или совокупности эталонных правил и характеристик. Корректность требует, чтобы программа была корректна на всех уровнях, начиная корректностью текста программы, до комплексной корректности, включающую в себя межмодульные связи, функциональность, динамичность.

Для программистов очень важно, чтобы программа была корректна, так как результат её работы – это результат работы программиста, показатель его навыков, опыта. Не так важно, чтобы приложение было быстрым, сколько чтобы оно работало так, как нужно.

**2.4. Тестируемость**

Тестируемость ПО – степень, в которой программная система или модуль поддерживает тестирование, то есть поиск неисправностей в системе. Тестируемость – очень важный фактор в программировании, так как она позволяет обеспечить другой важный фактор – корректность, и исправить не только текущие ошибки, но и исправить будущие.

Для программиста важно писать такие модули, которые будет легко тестировать, потому что на более поздних стадиях разработки, плохо протестированный модуль, может дать сбой в неординарных ситуациях, что повлечёт за собой глобальные изменения во всей программе, её архитектуре и логике.

Однако тестируемость – понятие, применимое не только к отдельным блокам, модулям в программе, но и ко всему приложению в целом. Для обеспечения тестируемости на ранних этапах разработки тестировщики создают специальные требования к программе, документацию, эталоны, которым обязаны соответствовать все модули программы. Такой строгий подход позволяет в будущем сэкономить огромное количество ресурсов, которые могли бы уйти на исправление множества ошибок и перестройку архитектуры приложения.

**2.5. Устойчивость**

Устойчивость – способность системы сохранять текущее состояние при влиянии внешних воздействий. Если текущее состояние при это не сохраняется, то такое состояние называется неустойчивым. Устойчивость ПО обычно обеспечивается с помощью введения различным форм избыточности, позволяющих иметь дублирующие модули программ, альтернативные программы для одних и тех же задач, осуществлять контроль над процессом исполнения программ.

Для пользователей важно, чтобы программа работала что бы они ни сделали. Программист должен рассчитывать на это и писать устойчивые программы, которые даже при неверных входных данных должны верно их обрабатывать, выводя ошибки или предупреждения.

**2.6. Стилизованность**

Стилизованность – принадлежность к одному стилю, схожесть определённых черт. Стилизованный код – код, написанный в одном стиле, по одним и тем же правилам, на протяжении всей программы.

В программировании очень часто приходится работать в команде, и часто приходится перечитывать ранее написанный код. В таком случае стилизованность – правило хорошего тона программирования. Когда программа написана в одном стиле, она проще, а значит быстрее, воспринимается и вызывает симпатию, так как просто красиво и структурированно выглядит.

Стилизованность проникает во все области программирования, начиная с одинакового стиля названия переменных, методов, классов, и заканчивая табуляцией и организацией файлов в решении.

**2.7. Читаемость**

Читаемость – характеристика, которая определяет, насколько легко вы можете понять некоторые части программы.

Читаемость – ещё одно правило хорошего тона. Читаемость позволяет человеку, не знакомому с программой, быстрее понять как она устроена и, например, помочь написавшему её в решении какой-то проблемы. На читаемость программы влияет множество факторов, начиная с опыта читающего и заканчивая стилем кода. Чтобы писать читаемый код, необходимо соблюдать табуляцию, модульность, стилизованность, декомпозицию, писать комментарии к коду, описывающие работу нетривиальных решений и не использовать чересчур сложных конструкций, таких как множество вложенных друг в друга методов или тернарных операторов.

**2.8. Универсальность**

Универсальность – возможность применения написанного кода в нескольких предметных областях, для решения различных задач.

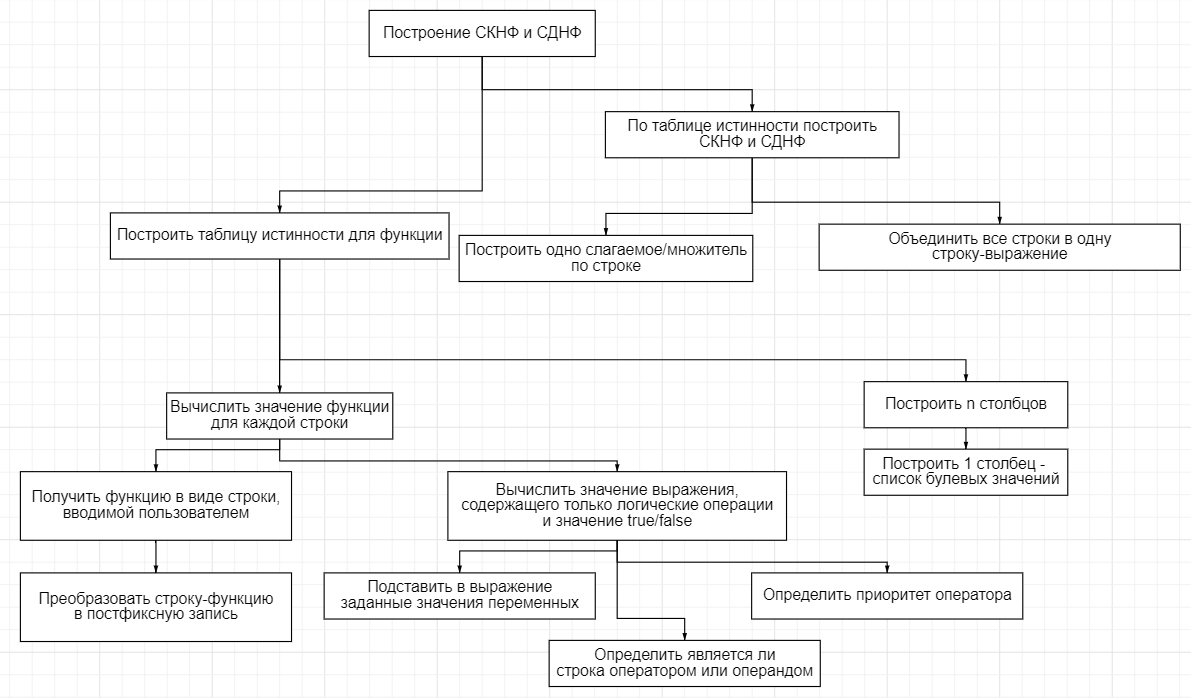
В программировании универсальность выражается сильнее всего в общедоступных библиотеках – сборников подпрограмм или объектов, которые используются для решения более конкретных задач. Это позволяет не только не использовать один и тот же код повторно, но и формализовать, унифицировать, стандартизировать программирование, чтобы все программисты общались на одном и том же языке.

# **3.Практика качественного программирования**

Рассмотрим применение вышеописанных приёмов на примере проекта «Логические функции». Его целью является построение совершенной дизъюнктивной и конъюнктивной нормальной формы логической функции.

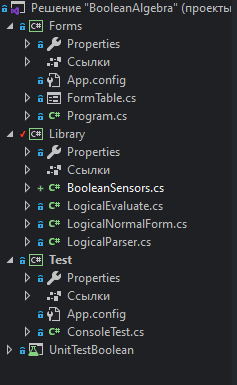
**3.1. Декомпозиция**

Для нашего проекта возможен следующий вариант декомпозиции:

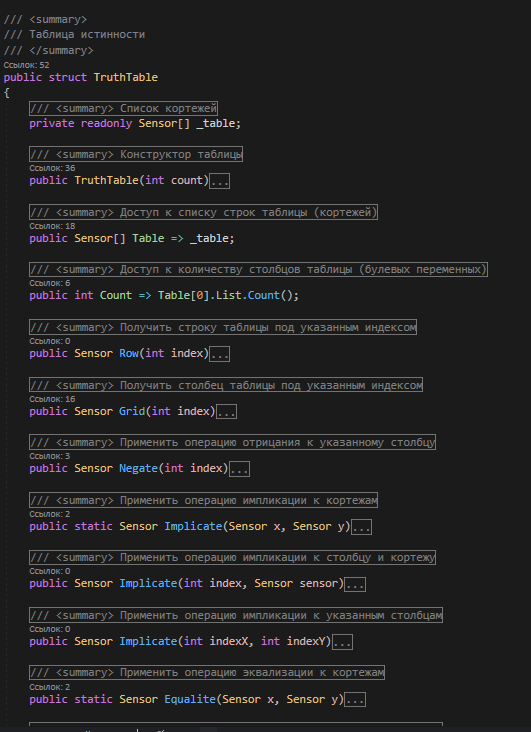


**3.2. Модульность**

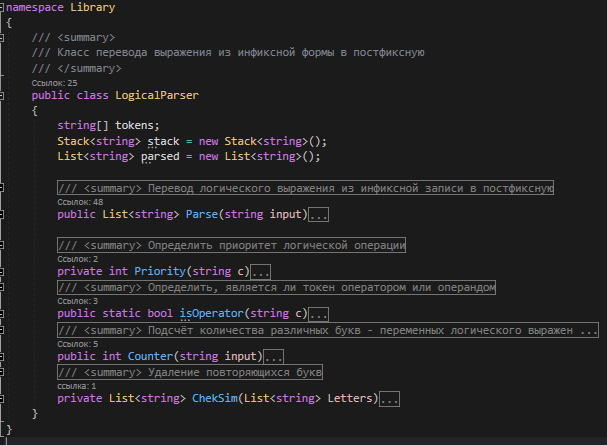
В нашем проекте роли модулей выполняют классы библиотеки классов.



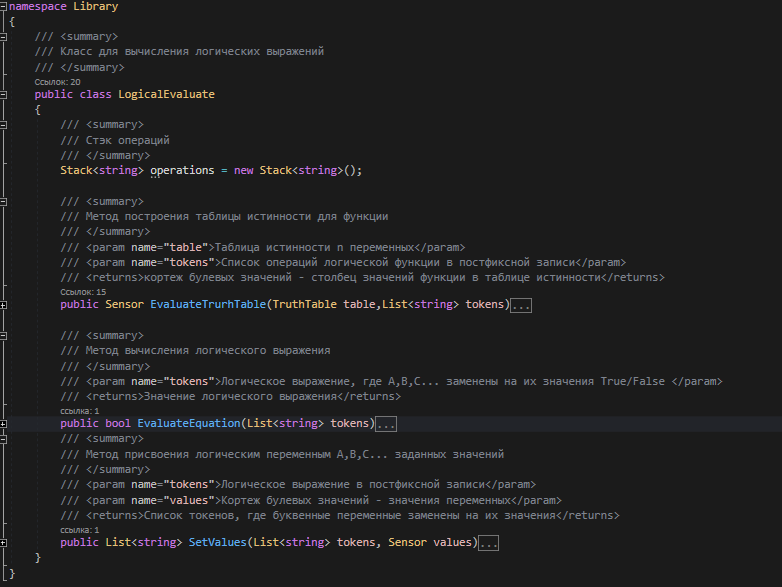
В файле BooleanSensors содержится две структуры: Sensor и TruthTable, которые представляют кортеж булевых значений и таблицу истинности соответственно. Sensor позволяет осуществлять логические операции над n логическими переменными. Основой для TruthTable служит массив типа Sensor, то есть массив кортежей булевых значений – таблицу истинности. Структура содержит методы для построения таблицы истинности n переменных и работы с ней (логические операции, получение строки или столбца по номеру).



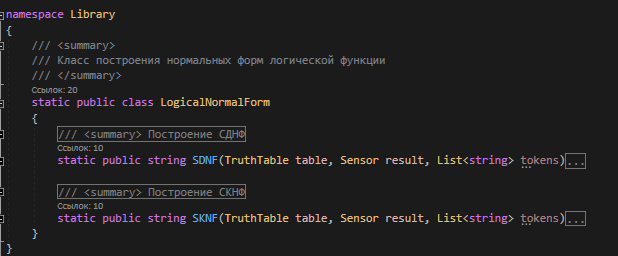
LogicalParser – Класс для перевода строки – логического выражения в постфиксную запись. Он содержит методы определения приоритета логической операции Priority(string c) и метод определения является ли строка оператором или операндом isOperator(string c). Главный метод этого класса – метод перевода логического выражения в постфиксную запись – Parse(string input).



Для вычисления значения логического выражения используется класс Logical Evaluate. В нём содержатся: метод установки булевых значений для переменных, метод расчёта выражения, в которой буквы-переменные заменены на их булевы значения и метод построения таблицы истинности для заданной функции – вычисление значения для каждой строки таблицы.

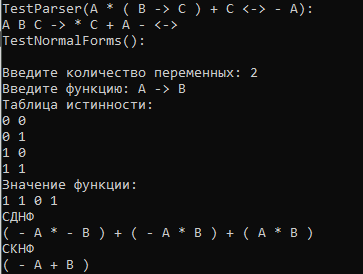


Класс LogicalNormalForm содержит методы для построения СНДФ и СКНФ по таблице истинности функции.



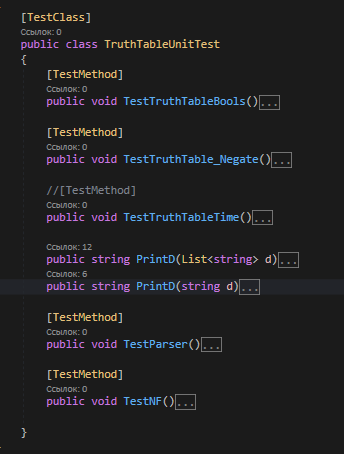
**3.3. Корректность**

Программа корректно переводит логическое выражения в постфиксную запись, строит таблицу истинности для заданной функции и нормальные формы. Действительно, импликацию можно представить как (не А или Б).



**3.4. Тестируемость**

Все методы приложения облагаются юнит-тестами и успешно их проходят



**3.5. Устойчивость**

Наше приложение также достаточно устойчива к изменениям в спецификации. Например, мы захотим вычислять значение не только логических, но и арифметических выражений. Нам необходимо будет лишь добавить новые операнды к методам Parse() и Evaluate(), чтобы программа могла работать с арифметическими выражениями.

**3.6. Стилизованность**

Код программы выдержан в одном стиле – поля классов названы в нижнем регистре, названия методов начинаются с большой буквы и написаны в стиле Camel, где несколько пишутся слитно и каждое новое с большой буквы.

**3.7. Читаемость**

Код программы обладает хорошей читаемостью. Он наполнен большим количеством комментариев, а поля и методы имеют говорящие названия. Также проект имеет прозрачную структуру и все методы идут последовательно в порядке их выполнения.

**3.8. Универсальность**

Классы нашей программы универсальны и могут быть использованы в других приложениях. Так, например, структуры Sensor и TruthTable подходят для любых видов работы с логическими функциями. Перевод логической функции в постфиксную запись может стать хорошей основой для интерпретатора выражений

# **4.Вывод**

Благодаря правилам качественного программирования, разработчикам программного обеспечения удаётся успешно проектировать и предоставлять пользователям действительно качественный продукт, который удовлетворяет потребностям не только крупных компаний и обычных пользователей, для которых важно быстродействие и удобство использования, но и таких же программистов, которые будут в дальнейшем заниматься развитием и поддержкой созданного приложения, для чего необходима точная организация и прозрачная архитектура проекта.