**программируемый контроллер** — цифровая электронная система, предназначенная для применения в производственной среде, которая использует программируемую память для внутреннего хранения ориентированных на потребителя [инструкций](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) по реализации таких специальных функций, как логика, установление последовательности, согласование по времени, счёт и арифметические действия для [контроля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C) посредством [цифрового](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) или [аналогового](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) ввода/вывода данных различных видов [машин](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) или процессов[[1]](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80#cite_note-1). Чаще всего ПЛК используют для [автоматизации технологических процессов](https://ru.wikipedia.org/wiki/%D0%90%D0%A1%D0%A3_%D0%A2%D0%9F). В качестве основного режима работы ПЛК выступает его длительное автономное использование, зачастую [в неблагоприятных условиях окружающей среды](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D1%89%D0%B8%D1%89%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), без серьёзного обслуживания и практически без вмешательства человека.

Иногда на ПЛК строятся системы [числового программного управления](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) станков.

ПЛК — устройства, предназначенные для работы в системах [реального времени](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F).

ПЛК имеют ряд особенностей, отличающих их от прочих электронных приборов, применяемых в промышленности:

* в отличие от [микроконтроллера](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80) (однокристального компьютера) — микросхемы, предназначенной для управления электронными устройствами — ПЛК являются самостоятельным устройством, а не отдельной микросхемой.
* в отличие от [компьютеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), ориентированных на принятие решений и управление оператором, ПЛК ориентированы на работу с машинами через развитый ввод сигналов [датчиков](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D1%87%D0%B8%D0%BA) и вывод сигналов на [исполнительные механизмы](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%B7%D0%BC);
* в отличие от [встраиваемых систем](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) ПЛК изготавливаются как самостоятельные изделия, отдельные от управляемого при его помощи оборудования.

Function Block Diagram (FBD) - это графический язык программирования, используемый в программируемых логических контроллерах (ПЛК) для разработки управляющих программ. В FBD программа представлена в виде схемы, состоящей из функциональных блоков и связей между ними.

Вот основные компоненты FBD:

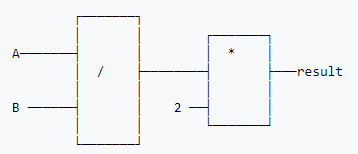
1. **Функциональные блоки (Function Blocks)**: Это основные строительные единицы программы в FBD. Каждый функциональный блок выполняет определенную операцию или функцию, такую как логические операции, арифметические вычисления, обработка сигналов и т.д. Функциональные блоки могут иметь различные входы и выходы, включая данные и управляющие сигналы.
2. **Связи (Connections)**: Связи используются для соединения выходов одного функционального блока с входами других блоков. Это позволяет передавать данные и управляющие сигналы между различными частями программы. Связи могут быть направленными или двунаправленными в зависимости от потребностей приложения.
3. **Константы и переменные**: Как и в других языках программирования, в FBD можно использовать константы и переменные для хранения данных и промежуточных результатов вычислений.
4. **Функции и операции**: FBD обычно предоставляет набор встроенных функций и операций для выполнения различных задач, таких как логические операции (AND, OR, NOT), арифметические вычисления, сравнения и т.д.

Программа образуется из списка цепей, выполняемых последовательно сверху вниз.

При программировании используются наборы библиотечных блоков и собственные блоки, также написанные на FBD или других языках.

Каждая отдельная цепь представляет собой выражение, составленное графически из отдельных элементов. К выходу блока подключается следующий блок, образуя цепь. Внутри цепи блоки выполняются строго в порядке их соединения. Результат вычисления цепи записывается во внутреннюю переменную либо подается на выход ПЛК.

Пример фрагмента программы на FBD: Переменную A разделить на переменную B, результат деления умножить на 2 и записать в переменную result:



Та же самая функция на псевдокоде: **result** := **2**\*(**A**/**B**);

**Преимущества функциональных блок-схем программирования ПЛК**

* **Гибкий визуальный редактор** | Редактор для программирования функциональных блок-схем очень удобен в использовании и обеспечивает простой способ создания любого макета.
* **Идеально подходит для сложных программных структур** | В лестничной логике пользователю придется использовать несколько ступеней для выполнения того, что возможно на одной странице FBD. Инструкции могут быть преобразованы непосредственно в сложные инструкции ПЛК, которые реализуют ПИД-контуры, управление движением и дополнительные инструкции (AOIs).
* **Удобный**| Визуальный редактор FBD понятен большинству пользователей. Схема процесса может быть воссоздана с помощью метода перетаскивания, который практически не оставляет места для догадок.

**Недостатки структурированного текста**

* **Трудно стандартизировать** | Из-за гибкости компоновки сложно стандартизировать программы, написанные на FBD. У каждого программиста ПЛК будет свой подход, отличный от других. Тем, кто отстает, будет трудно разобраться в потоке информации. **‍**
* **Проблемы в масштабе** | FBD великолепен, когда дело доходит до небольших реализаций конкретных областей процесса. Однако по мере усложнения программы легко заблудиться во всех этих таблицах.

Structured Text (ST) - это текстовый язык программирования, используемый в программируемых логических контроллерах (ПЛК) для разработки управляющих программ. ST основан на синтаксисе языка программирования Pascal и предоставляет более высокий уровень абстракции по сравнению с графическими языками, такими как Ladder Diagram (LD) или Function Block Diagram (FBD).

Вот основные черты и компоненты ST:

1. **Структура программы**: Программа на ST состоит из последовательности инструкций, объединенных в блоки и процедуры. Это позволяет создавать структурированный и модульный код, что облегчает его понимание и обслуживание.
2. **Поддержка типов данных**: ST поддерживает различные типы данных, включая целые числа, вещественные числа, строки, массивы и структуры данных. Это позволяет программистам работать с данными разного типа и сложности.
3. **Операторы и функции**: ST предоставляет широкий набор операторов и функций для выполнения различных операций, включая арифметические вычисления, логические операции, сравнения, циклы и условные операторы.
4. **Обработка событий**: В ST можно организовать обработку событий и реагирование на изменения внешних условий, таких как сигналы ввода-вывода, по времени или по другим событиям.

Пример: вычисление максимума из массива  
VAR\_CONSTANT

Array\_Sz: BYTE := 4;

END\_VAR

VAR

Iter: BYTE;

arr: ARRAY [1..Array\_Sz] of real:=3.2, 4.2 ,1.4, 7.8;

fnd\_max:REAL := -1.2E38;

END\_VAR

FOR Iter := 1 TO Array\_Sz DO

fnd\_max := MAX(fnd\_max, arr[Iter]);

END\_FOR

**Преимущества структурированного текстового программирования ПЛК**

* **Интуитивно понятен для других языков программирования** | Как упоминалось выше, [Структурированный текст](https://www.solisplc.com/tutorials/structured-text-logic-and-mathematical-instructions-motor-starter-interview-practice) прост в освоении для тех, кто хочет перейти из сферы разработки программного обеспечения. Он имеет те же структуры, парадигмы программирования и функции, которые можно было бы ожидать увидеть в C или Java.
* **Высокая сложность** | Структурированный текст обеспечивает большую гибкость по сравнению с другими языками и, таким образом, упрощает реализацию расширенных функциональных возможностей для тех, кто владеет языком.
* **Возможность переноса** | Структурированный текст стандартизирован в большинстве систем ПЛК, что упрощает миграцию между платформами. Вы обнаружите существенные различия в других языках между платформами, однако структурированный текст может быть реализован на аппаратных и программных платформах.

**Недостатки структурированного текста**

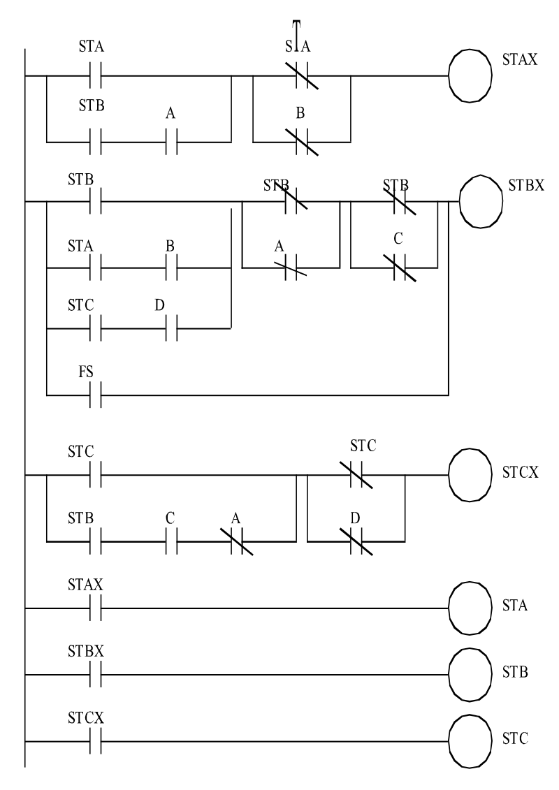
* **Сложно устранять неполадки** | По сравнению с программированием на лестничной логике структурированный текст намного сложнее с точки зрения устранения неполадок. Здесь нет визуальных очередей, меньше наглядных пособий и, как правило, больше кода в одной строке. Тем, кто не знаком с этим языком, будет трудно разобраться в технологическом процессе.
* **Подвержены ошибкам** | Структурированный текст обеспечивает большую гибкость для пользователя. Однако эта гибкость достигается за счет стандартизации. Пользователи должны использовать лучшие практики разработки программного обеспечения для создания безопасных резервных версий и устранения любых потенциальных сбоев программного обеспечения.

Ladder Diagram (LD) - это графический язык программирования, используемый в промышленных логических контроллерах (ПЛК) для разработки управляющих программ. LD моделирует логику реле и электрические схемы. Синтаксис языка удобен для замены логических схем, выполненных на релейной технике. Ориентирован на инженеров по автоматизации, работающих на промышленных предприятиях. Обеспечивает наглядный интерфейс логики работы контроллера, облегчающий не только задачи собственно программирования и ввода в эксплуатацию, но и быстрый поиск неполадок в подключаемом к контроллеру оборудовании.

Программа на языке релейной логики имеет наглядный и интуитивно понятный инженерам-электрикам графический интерфейс, представляющий логические операции как электрическую цепь с замкнутыми и разомкнутыми контактами. Протекание или отсутствие тока в этой цепи соответствует результату логической операции (истина — если ток течёт; ложь — если ток не течёт).

Основные компоненты LD включают:

1. **Контакты (Contacts)**: Представляют собой условия или сигналы, которые могут быть либо открытыми (недействующими), либо закрытыми (действующими). Контакты могут представлять входные сигналы, состояния устройств или результаты логических операций.
2. **Катушки (Coils)**: Представляют собой действия, которые должны быть выполнены в зависимости от состояний контактов. Катушки обычно представляют выходные сигналы, управляющие действиями, такими как включение или выключение устройств.
3. **Реле (Relays)**: Это особые элементы, которые позволяют создавать условные логические операторы, такие как AND, OR, NOT, а также временные задержки и другие функции.
4. **Сети (Networks)**: Сети - это группы контактов, катушек и реле, объединенные вместе для выполнения определенной логической функции. Сети могут представлять собой последовательные или параллельные соединения элементов.



**Преимущества программирования ПЛК с лестничной логикой**

* Простота реализации и устранения неполадок | Лестничная логика - это визуальный язык, который обеспечивает подтверждение состояния большинства инструкций. Другими словами, человеку, слабо разбирающемуся в конкретном процессе, легко ознакомиться с программой и понять логику.
* Модульная конструкция | логическая схема лестницы может быть легко изменена путем добавления или вычитания логики. Каждая ступенька является отдельным условием и может быть удалена или добавлена по мере необходимости.
* Устойчивость и согласованность | лестничная логика позволяет пользователю реализовывать множество функций. Однако язык сильно стандартизирован и не обеспечивает полной гибкости, что позволяет поддерживать согласованность кода в различных реализациях.

**Недостатки лестничной логики**

* Крутая кривая обучения | Лестничная логика - простой язык, но не очень понятный для тех, кто имеет опыт работы с C, C ++, Java или Python. Как уже было сказано, инженерам-электрикам и тем, кто обладает базовыми знаниями в области программирования сборок, возможно, будет легче их усвоить.
* Медленное развертывание | Из-за визуального характера лестничной логики программисту требуется больше времени, чтобы создать логику, которую он задумал. Необходимо перетаскивать элементы, которые замедляют процесс разработки по сравнению с другими современными языками программирования.
* Не интуитивно для сложных приложений | Лестничная логика великолепна, когда дело доходит до последовательных логических задач. Однако, когда дело доходит до современной теории управления, которая включает в себя ПИДЫ, регулирование потока, аналоговые датчики и контуры обратной связи, это не всегда легко реализовать и расшифровать.

**где используются**

Задачи: FBD хорошо подходит для задач, где требуется параллельная обработка различных сигналов и выполнение различных логических операций. Это может включать в себя управление системами манипуляции, обработку данных и управление сенсорами.

Задачи: ST подходит для сложных вычислений, обработки данных и реализации алгоритмов. Он также хорошо подходит для задач, требующих структурного программирования, таких как управление логикой систем и обработка больших объемов данных.

Задачи: LD хорошо подходит для простых логических операций, управления включением и выключением устройств, а также для задач, связанных с управлением вводом-выводом. Он часто используется для автоматизации промышленных процессов, управления системами безопасности и управления оборудованием.