4. PLC 프로그래밍 언어

4.1 국제 표준 IEC 61131-3 소개

(PLC 프로그래밍 언어 – 표준 이해의 출발점)

✅ 개요

IEC 61131-3은 **국제전기표준회의(IEC)**가 제정한
PLC 프로그래밍 언어와 구조에 대한 세계 공통의 표준이다.

📌 이 표준은 **PLC 제조사 간의 언어 및 구조 차이를 통일**하고,
프로그램 이식성과 유지보수성을 향상시키기 위해 만들어졌다.

✅ IEC 61131 표준 구성 요약

파트	설명
IEC 61131-1	PLC 일반 개념 및 용어
IEC 61131-2	하드웨어 요구사항 및 전기적 특성
IEC 61131-3	⭐ 프로그래밍 언어 및 소프트웨어 모델 정의
IEC 61131-4	사용자 가이드라인
IEC 61131-5	통신 서비스 정의

✅ IEC 61131-3의 핵심 내용

1. **PLC 프로그래밍 언어 5가지 정의**
2. **소프트웨어 구조 모델 (Program, Task, Function, Function Block)**
3. **데이터 타입, 변수 선언 방법**
4. **모듈화, 재사용성, 계층적 구조 지원**
5. **공통 문법과 의미 부여 → 제조사 간 호환성 증가**

✅ IEC 61131-3이 정의하는 5가지 언어

언어	약어	설명
래더 다이어그램	LD	릴레이 회로 유사, 유지보수 용이한 대표 언어
명령어 리스트	IL *(※ 폐지됨)*	어셈블리 유사한 텍스트 언어 (v3부터 제거됨)
구조적 텍스트	ST	Pascal 기반 고급 텍스트 언어, 복잡한 연산에 적합

언어	약어	설명
펑션 블록 다이어그램	FBD	블록 기반 시각적 표현, 논리 연결 쉬움
시퀀셜 펑션 차트	SFC	단계(스텝) + 전이(트랜지션) 기반 시퀀스 제어

✅ 소프트웨어 구성 요소

요소	설명
Program	PLC 실행 단위 전체 로직
Task	주기/이벤트 기반 실행 제어
Function Block (FB)	변수 + 내부 로직 포함 블록 (재사용 가능)
Function (FC)	입력 → 출력 단일 연산 함수
Global/Local Variables	전역/지역 변수의 구분과 선언 방식 존재

✅ IEC 61131-3의 장점

항목	설명
제조사 독립성	동일한 논리 구조를 여러 제조사에서 적용 가능
이식성 향상	코드의 재사용 및 장비 이전 간소화
모듈화 구조	블록 기반 설계로 유지보수 용이
고급 기능 제공	조건 분기, 반복문, 수학 연산 등 고급 제어 가능 (ST 등 활용 시)
표준 문서화	프로젝트 문서화와 인증 대응 용이 (ISO, CE 등)

✅ 실제 현장 적용 예

기업	적용 예시
Siemens	TIA Portal에서 LD, FBD, ST, SFC 지원
Mitsubishi	GX Works3에서 LD, FBD, ST 일부 지원
Omron	Sysmac Studio에서 IEC 기반 언어 모두 지원
Beckhoff (TwinCAT)	ST 기반 개발 중심
Rockwell (Allen-Bradley)	Studio5000에서 LD, FBD, ST 등 구현

✅ 정리

- **IEC 61131-3은 전 세계 PLC 프로그래밍의 공통 기준**이다.
- 래더 다이어그램만이 아닌 **FBD, ST, SFC 등 다양한 언어를 통합적으로 설계할 수 있는 기반**을 제공한다.
- 현대 PLC 시스템은 이 표준을 기반으로, **복잡한 공정을 모듈화하고 구조화**된 방식으로 제어하는 데에 중점을 둔다.

4.2 LD (Ladder Diagram, 래더 다이어그램)

(IEC 61131-3 언어 – 가장 널리 쓰이는 PLC 언어)

✅ 개요

래더 다이어그램(LD, Ladder Diagram)은
전통적인 **릴레이 논리 회로**의 형식을 본떠 만든 **그래픽 기반의 PLC 프로그래밍 언어**다.

📌 전기 기술자나 유지보수 담당자가 **직관적으로 이해하고 디버깅하기 쉬운 구조**이기 때문에,
대부분의 PLC 제조사와 산업 현장에서 가장 많이 사용된다.

✅ LD의 구조와 구성 요소

📌 기본 형식

래더 다이어그램은 **좌측 전원(Rail) → 우측 전원(Rail)**으로 **전류가 흐르는 형태**로 구성된다.

```
|----[ ]----[ ]--------------------( )----|
     X0     X1                        Y0
```

구성	설명
`[ ]`	입력 접점 (X0, X1 등)
`( )`	출력 코일 (Y0 등)
`--`	전선 역할 (논리 흐름)

📌 주요 기호 의미

기호	이름	기능
`[ ]`	**NO 접점 (Normally Open)**	ON일 때 전류 통과
`[/]`	**NC 접점 (Normally Closed)**	OFF일 때 전류 통과
`( )`	**출력 코일**	출력 신호 발생
`SET/RES`	Set/Reset 코일	유지 출력 (SR 플립플롭)
`OUT`	일반 출력 (Y, M 등)	출력 상태 변경

기호	이름	기능
`T`, `C`	타이머/카운터 코일	조건에 따라 작동 시간/횟수 제어

✅ LD 기본 예제

📌 예제 1: 두 스위치를 모두 눌러야 출력이 켜짐 (AND 논리)

```
|----[ X0 ]----[ X1 ]----------------( Y0 )----|
```

→ X0과 X1이 모두 ON일 때만 Y0 출력

📌 예제 2: 둘 중 하나라도 눌리면 출력 켜짐 (OR 논리)

```
|----[ X0 ]----------------( Y0 )----|
|----[ X1 ]----------------|
```

→ X0 또는 X1 중 하나라도 ON이면 Y0 출력

✅ LD의 핵심 제어 요소

기능	기호/명령	설명
자기 유지 회로	`[ ]` → `( )` → `[ M ]` → 다시 `[ ]`	ON 유지
타이머	`T0` / `TON`, `TOF`	지연 또는 유지 시간 설정
카운터	`C0` / `CTU`, `CTD`	펄스 횟수 누적 후 동작
인터록	`[/ M1 ]`	다른 조건에 의해 차단
조건 분기	`IF` 구조 없이 접점으로 구현	조건식 조합 가능
Set/Reset	`SET`, `RST`	ON/OFF 상태 유지 제어

✅ LD vs 논리 회로 비교

논리 회로	래더 다이어그램
A AND B → Y	`[ A ] -- [ B ] → ( Y )`
A OR B → Y	`[ A ] → ( Y )`, `[ B ] → ( Y )`
A AND (NOT B) → Y	`[ A ] -- [/ B ] → ( Y )`

✅ LD의 장점과 한계

장점	설명
직관적	릴레이 회로 유사 → 전기 기술자 이해 쉬움
표준화	거의 모든 PLC에서 지원 (IEC 61131-3)
디버깅 용이	시각적으로 흐름 확인 가능
모든 기능 커버 가능	타이머, 카운터, 연산, 비교도 표현 가능

한계	설명
복잡한 연산 불리	수학/문자열 처리에 부적합 (→ ST 추천)
가독성 저하	로직이 커질수록 보기 어려움
함수화 어려움	코드 재사용성은 다른 언어보다 낮음

✅ 실무 적용 예시

예제	설명
비상정지 스위치 → 출력 OFF	`[ /X0 ] → ( Y0 )`
자기 유지 회로 (토글)	`[ X1 ] → ( M0 )`, `[ M0 ] → ( Y0 )`
시퀀스 제어 (A → B → C)	`[ SensorA ] → ( YB )`, `[ SensorB ] → ( YC )` 등
경광등 점멸	`[ T0.DN ] → ( Y_Lamp )` with `TON` 타이머

✅ 정리

- **LD는 유지보수성과 가독성이 뛰어난 PLC 기본 언어**다.
- 단순한 논리부터 중간 수준의 시퀀스 제어까지 적합하며,
  타이머/카운터/인터록/조건 분기 모두 표현 가능하다.
- 복잡한 수식/반복 처리는 다른 언어(FBD, ST 등)와 병행 사용하는 것이 효과적이다.

4.3 STL (Instruction List, 명령어 리스트)

(IEC 61131-3 언어 – 어셈블리 스타일의 텍스트 기반 언어)

✅ 개요

STL (Instruction List)은 **PLC 프로그래밍을 위해 정의된 저수준 텍스트 언어**로,
어셈블리 언어처럼 **명령어 단위로 로직을 기술**한다.

📌 STL은 간결하고 CPU와 가까운 명령어 기반의 언어이지만,
이해하기 어려워 유지보수성이 낮다는 단점 때문에
IEC 61131-3 **버전 3 이후로는 공식 폐지되었으며**,
일부 제조사에서만 **하위 호환성 목적으로 유지**되고 있다.

✅ STL의 기본 구조

STL은 **명령어 + 피연산자** 형식으로 구성된다.

```
LD      X0          ; X0 입력을 누적 스택에 적재
AND     X1          ; AND 연산 수행
OUT     Y0          ; 결과를 Y0 출력으로 설정
```

- `LD`: Load – 입력을 메모리에 적재
- `AND`, `OR`, `NOT`: 논리 연산
- `OUT`: 출력에 결과 전달
- `=`, `:=`: 결과 할당

✅ 주요 명령어 요약

명령어	의미	설명
`LD`	Load	피연산자 로드 (시작점)
`AND`, `OR`, `XOR`	논리 연산	누적값과 연산
`NOT`	부정	현재 누적된 값 반전
`OUT` / `ST`	Output / Store	연산 결과를 출력 또는 변수에 저장
`A`, `O`, `=`, `:=`	약식 표현	일부 제조사에서는 단축 표현 허용
`LDN`	Load Not	부정된 입력을 로드
`ANDN`, `ORN`	부정 연산	NOT X0 같은 표현
`JC`, `JMP`	Jump Conditional	조건 분기 (라벨 기반)
`CALL`	서브루틴 호출	Function/Function Block 실행
`TON`, `TOF`, `TP`	타이머 명령어	시간 제어

✅ STL 예제 1: AND 조건 제어

```
LD      X0
AND     X1
OUT     Y0
```

→ X0과 X1이 모두 ON이면 Y0 ON

✅ STL 예제 2: OR 조건 제어

```
LD      X0
OR      X1
OUT     Y0
```

→ X0 또는 X1이 ON이면 Y0 ON

✅ STL 예제 3: 비상 정지 조건 포함

```
LD      X0
ANDN    E_STOP      ; E_STOP이 OFF일 때만 동작
OUT     MOTOR_ON
```

✅ STL vs LD 비교

기능	LD 표현	STL 표현
AND 제어	`[X0]--[X1]--(Y0)`	`LD X0 / AND X1 / OUT Y0`
OR 제어	`[X0]--(Y0)` `[X1]--(Y0)`	`LD X0 / OR X1 / OUT Y0`
비상 정지 포함	`[ /E_STOP ]`	`ANDN E_STOP`

✅ 장점과 단점

장점	설명
간결성	단일 라인으로 빠르게 로직 구성 가능
처리 속도 빠름	중간 추상화 없음, CPU 직접 처리
다른 언어와 유사함	어셈블리, 저수준 언어 익숙한 개발자에 유리

단점	설명
가독성 낮음	유지보수, 교육, 협업에 불리

단점	설명
시각적 디버깅 불가	LD, FBD처럼 흐름을 눈으로 파악 불가
표준상 폐지됨	IEC 61131-3 v3 이후 공식 제거됨 (→ ST 권장됨)

✅ 제조사별 지원 여부

제조사	STL 지원 여부
Siemens	지원 (STEP 7, TIA Portal - SCL 권장)
Mitsubishi	일부 지원 (IL 형식으로 제한적)
LS산전	제한적 (XG5000에서 IL 코드 불완전 지원)
Omron	구형 PLC에서만 일부 지원
Allen-Bradley	별도 STL 미제공, ST 또는 LD 위주

✅ STL 사용 시 주의사항

- **신규 프로젝트에서는 ST 또는 LD 사용 권장**
- STL은 **기존 레거시 시스템 유지보수용**으로만 선택
- 코드를 작성할 경우 **주석 필수**, 블록 나누기 필수

✅ 정리

- STL은 **텍스트 기반 저수준 명령어 언어**로, **LD보다 빠르지만 가독성이 낮다**
- **단순한 논리 연산이나 빠른 코드 작성에는 유리하지만,**
  현대적 유지보수, 문서화, 협업에는 부적합
- IEC 61131-3 v3 이후로는 **ST(Structured Text)**가 STL을 대체하며 권장된다.

4.4 FBD (Function Block Diagram)

(IEC 61131-3 언어 – 시각적 데이터 흐름 중심 제어 언어)

✅ 개요

FBD(Function Block Diagram)는
기능 블록(Function Block)들을 **그래픽 상에서 배치하고 선으로 연결하여 로직을 표현하는 언어**다.

📌 전기 회로처럼 보이는 블록들을 조합하여 **입력 → 블록 → 출력**의 데이터 흐름을 시각적으로 표현하기 때문에
제어 논리, 신호 흐름, 조건 판단을 직관적으로 구성할 수 있다.

✅ FBD 구성 요소

구성 요소	설명
Function Block	특정 기능을 수행하는 논리 블록 (예: AND, OR, Timer 등)
Input/Output Port	블록의 입력값과 출력값 인터페이스
Wire (선)	블록 간의 연결 선 – 신호 흐름을 표현
Constant (상수)	고정값을 직접 연결
Variable (변수)	메모리 공간에 저장되는 값, 입력/출력으로 사용

✅ 기본 FBD 예시

📌 예제: X0 AND X1 → Y0

```
X0 ─┐
    ├─ AND ──> Y0
X1 ─┘
```

➡️ X0, X1이 모두 ON일 때 Y0 출력

✅ 주요 블록 유형

블록 종류	설명	예시
논리 연산	AND, OR, NOT, XOR	조건 판단
타이머	TON, TOF, TP	시간 기반 제어
카운터	CTU, CTD, CTUD	입력 펄스 누적/감소
수학 연산	ADD, SUB, MUL, DIV	연산식 구현
비교 연산	GT, LT, EQ, NE	값 비교 후 조건 출력
MUX/SEL	멀티플렉서, 선택기	조건에 따라 분기
MOVE	값 복사	입력값 → 출력값으로 전송
Set/Reset	상태 유지	래치 동작 구현
Function Block (FB)	사용자 정의 로직	자체 블록 정의 가능

✅ 예제: 타이머(TON)와 램프 출력

```
X0 ──> TON ──> Y_LAMP

TON 파라미터:
IN: X0
PT: T#3s
Q: 램프 출력 (Y_LAMP)
ET: 경과 시간 (T 변수에 저장 가능)
```

➡️ X0 ON 상태가 3초 유지되면 Y_LAMP ON

✅ 예제: 카운터 + 비교

```
Pulse ──> CTU ──> CNT
CNT.Q ──> EQ ──> Done (M10)
              └── CntValue == 10
```

➡️ 펄스가 10번 입력되면 M10 ON

✅ FBD의 장점

항목	설명
시각적 표현	블록 간 흐름을 쉽게 파악 가능
재사용성	FB 블록을 재활용하여 모듈 설계 용이
디버깅 편리	신호 흐름 추적이 직관적
기초 교육에 적합	초보자도 쉽게 구조 이해 가능

✅ FBD의 단점

항목	설명
복잡한 조건 표현 어려움	중첩된 분기나 루프 제어는 부적합
수학/문자열 처리에 비효율적	복잡한 계산은 ST 권장
대규모 로직에선 난잡해짐	화면 구성 한계 → 구조 분해 필요
배선 오류 가능성	연결선 겹침 등으로 논리 오류 유발 우려 있음

✅ 제조사 지원 예

제조사	FBD 지원 여부
Siemens	TIA Portal에서 FBD 완전 지원
Mitsubishi	GX Works3에서 일부 FBD 지원
Beckhoff	TwinCAT에서 고급 FBD 사용 가능
Omron	Sysmac Studio에서 FBD 통합 지원
Allen-Bradley	Studio 5000 FBD 완전 지원

✅ FBD vs LD vs ST 비교

항목	LD (래더)	FBD	ST
표현 방식	릴레이 회로	블록 간 연결	텍스트 기반
시각성	높음	매우 높음	낮음
복잡한 연산	어려움	제한적	유리
수학 계산	비효율	보통	우수
초보자용	매우 적합	적합	어렵다
대규모 로직	분할 필요	블록 분리 필요	가장 유리

✅ 정리

- **FBD는 블록 단위의 그래픽 제어 언어로,**
  입력 → 처리 → 출력의 데이터 흐름을 시각적으로 표현한다.
- 타이머, 비교기, 연산기 등 다양한 블록을 연결해 **로직을 시각적으로 구현**하며,
  초보자부터 중급 수준의 제어까지 널리 활용된다.
- 복잡한 연산, 반복 제어는 **ST(Structured Text)**와 병행해서 사용하는 것이 일반적이다.

4.5 SFC (Sequential Function Chart)

(IEC 61131-3 언어 – 상태 기반 순차 제어의 표준 표현)

✅ 개요

SFC(Sequential Function Chart)는
공정이나 기계의 상태 전이를 "단계(Step)"와 "전이(Transition)"로 표현하는 그래픽 기반 언어다.

📌 시퀀스 기반의 작업 순서를 명확히 표현할 수 있어
자동화 장비, 로봇 제어, 조립 공정 등 상태 기반 제어에 특히 적합하다.

✅ 핵심 개념

구성 요소	설명
Step (단계)	동작의 상태를 의미. 활성화되면 지정된 작업 수행
Transition (전이)	다음 단계로 넘어가기 위한 조건
Action (행동)	Step이 활성화될 때 실행되는 연산 또는 출력
Flow Line (흐름선)	Step과 Transition을 연결하는 선

✅ 기본 SFC 구조

```
[Step_1] —> (Transition_1) —> [Step_2] —> (Transition_2) —> [Step_3]
```

- Step은 각각 하나의 동작 상태
- Transition은 그 상태를 넘어갈 조건
- 각 Step에 Action을 붙여 제어 동작 수행

✅ 예제: 모터 시퀀스 (시작 → 동작 → 정지)

```
[Idle]
   ↓ (Start == TRUE)
[Running]
   ↓ (Timer >= 10s)
[Stopping]
   ↓ (MotorSpeed == 0)
[Idle]
```

Step	Action
Idle	아무 동작 없음
Running	Motor ON
Stopping	Motor OFF

➡️ 전형적인 자동 제어 시퀀스 구조를 명확하게 표현

✅ SFC의 동작 규칙

항목	설명
초기 Step 지정	프로그램 시작 시 활성화되는 Step
병렬 분기/병합	여러 Step을 병렬로 실행하거나 합치는 구조 가능
조건 분기	하나의 Transition에서 여러 방향으로 분기
Loop 구성 가능	Step 간 연결을 통해 반복 시퀀스 구현
비동기 전이 가능	타이머/센서/입력값에 따라 비순차 전이 가능

✅ SFC vs 상태도 vs 플로우차트 비교

항목	SFC	상태도(State Machine)	플로우차트
단위	Step + Transition	State + Event	Action 흐름
논리 구성	모듈화된 단계 구조	이벤트 중심 상태 변화	절차적 처리 흐름
표현 목적	시퀀스 제어	복잡한 논리 전이	순차 작업 흐름
IEC 표준	✅ IEC 61131-3	❌ 비표준	❌ 비표준

✅ 실무에서의 SFC 활용 예

분야	활용 예
제조 공정	부품 투입 → 조립 → 검사 → 분류
로봇 제어	대기 → 이동 → 집기 → 배치
포장 기계	제품 감지 → 포장지 내림 → 씰링 → 배출
HVAC	모드 설정 → 팬 작동 → 온도 조절 → 정지

✅ SFC의 장점

항목	설명
시각적 명료성	공정 흐름을 단계적으로 표현 가능
모듈화 쉬움	Step 단위로 기능 분리하여 유지보수 유리
디버깅 용이	현재 활성화된 Step 확인 가능
복잡한 순차 제어에 특화	이벤트 기반보다 더 안정적 시퀀스 구성 가능

항목	설명
LD/FBD/ST와 병용 가능	Step 내 Action은 다양한 언어로 작성 가능

✅ SFC의 단점

항목	설명
단순 조건 판단에는 과함	간단한 논리는 LD나 ST가 더 효율적
지나치게 복잡한 흐름은 난잡해짐	분기, 병렬이 많을수록 복잡해짐
초기 학습 장벽	상태기반 로직에 익숙하지 않으면 해석 어려움

✅ 제조사별 지원 여부

제조사	SFC 지원
Siemens	TIA Portal (Graph) 완전 지원
Mitsubishi	GX Works3에서 일부 지원
Omron	Sysmac Studio에서 SFC 지원
Beckhoff	TwinCAT에서 SFC 완전 지원
Allen-Bradley	Studio 5000 일부 지원 (RSLogix의 SFC 스타일)

✅ LD/FBD/ST와의 연동

SFC는 **Step을 정의하는 메타 구조**이고,
각 Step 안의 Action을 실제 제어 언어로 작성한다.

Step 안의 Action 표현	사용 가능 언어
래더로 제어	LD
수식으로 제어	ST
블록 연산	FBD

✅ 정리

- **SFC는 공정 제어, 기계 시퀀스 구현에 최적화된 상태 기반 언어**
- Step/Transition 구조로 공정 흐름을 시각적으로 표현하고
  Action은 LD/FBD/ST 등으로 구성해 유연한 설계 가능
- 특히 **다단계 시퀀스, 공정 자동화, 로봇 동작 루틴 등**에 널리 사용됨

4.6 ST (Structured Text)

(IEC 61131-3 표준 언어 – 고급 로직 처리를 위한 텍스트 기반 언어)

✅ 개요

ST(Structured Text)는
PLC 프로그래밍 언어 중 **가장 강력하고 유연한 고급 언어**로,
Pascal 계열의 문법을 기반으로 만들어진 **텍스트 기반 언어**이다.

📌 수치 연산, 문자열 처리, 배열, 조건문, 반복문, 함수 호출 등
복잡한 논리와 알고리즘 구현에 최적화된 PLC 언어로,
복잡한 계산, 데이터 처리, 사용자 정의 함수 등에서 뛰어난 표현력을 가진다.

✅ ST의 문법 구조

ST는 일반적인 고급 언어처럼 **절차형 문장, 조건문, 반복문, 함수 호출**로 구성된다.

```
IF Sensor1 = TRUE AND Temp > 30 THEN
    Fan := TRUE;
ELSE
    Fan := FALSE;
END_IF;
```

✅ 주요 문법 요소

문법	설명	예시
변수 대입	`:=` 사용	`Motor := TRUE;`
조건문	IF ~ THEN ~ ELSE ~ END_IF	`IF A = B THEN ...`
반복문	FOR, WHILE, REPEAT	`FOR i := 1 TO 10 DO ...`
논리 연산자	AND, OR, NOT	`IF A AND NOT B THEN ...`
비교 연산자	`=`, `<>`, `<`, `>`, `<=`, `>=`	`IF X >= 100 THEN ...`
수학 연산자	`+`, `-`, `*`, `/`, `MOD`	`Z := X * Y + 3;`
타이머 제어	`TON`, `TOF`, `TP`	`Timer(IN := Start, PT := T#5s);`
함수 호출	사용자 정의/라이브러리 함수 호출	`Result := AVG(ValueArray);`

✅ ST 기본 예제

📌 예제 1: 단순 온도 제어

```
IF Temp > 25.0 THEN
    Cooler := TRUE;
ELSE
    Cooler := FALSE;
END_IF;
```

📌 예제 2: 반복문을 활용한 평균 계산

```
Sum := 0;
FOR i := 1 TO 10 DO
    Sum := Sum + Sensor[i];
END_FOR;
Average := Sum / 10;
```

📌 예제 3: 타이머를 이용한 펌프 제어

```
PumpTimer(IN := Start, PT := T#10s);
IF PumpTimer.Q THEN
    Pump := FALSE;
END_IF;
```

✅ 고급 기능

기능	설명
배열 사용	`Sensor: ARRAY[1..10] OF REAL;`
구조체 사용	`TYPE Status : STRUCT ... END_STRUCT`
라이브러리 함수 활용	`SIN()`, `SQRT()`, `LIMIT()` 등 수학 함수
유저 정의 함수/FB	`FUNCTION`, `FUNCTION_BLOCK` 정의 가능
파일/네트워크 처리	일부 PLC 플랫폼에서 외부 데이터 핸들링 가능
문자열 처리	`CONCAT()`, `LEFT()`, `LEN()` 등 사용 가능

✅ ST의 장점

항목	설명
복잡한 로직 구현에 최적	수식, 반복, 조건 분기 등 표현력 우수
코드 재사용 용이	함수, FB, 모듈화 구성 가능

항목	설명
가독성/문서화 용이	고급 언어와 유사한 구조
자동화 설계 최적화	고급 기능 + 유연한 제어 가능
PLC 외 장비와 연계 쉬움	JSON, 문자열, 수학, 통신 연산 가능

✅ ST의 단점

항목	설명
전기 기술자에겐 난해	비전공자에겐 코드 이해 어려움
시각적 디버깅 불가	LD/FBD처럼 흐름을 눈으로 보기 어려움
간단 제어엔 과함	릴레이 논리나 간단 ON/OFF 제어는 LD/FBD가 더 효율적

✅ ST 사용이 유리한 경우

상황	이유
복잡한 연산식 필요	LD/FBD로 표현이 매우 번거로움
다중 조건 분기	`IF ~ ELSIF ~ ELSE` 처리 용이
반복문, 배열 처리	센서 다중 제어, 평균값 계산 등
사용자 정의 기능	재사용 가능한 Function Block 설계
고급 통신/데이터 연산	Modbus, JSON 문자열 처리, 수학 연산

✅ 실제 산업 적용 예

분야	활용 예
HVAC 제어	온도, 습도, 풍속 계산 제어
에너지 모니터링	전류/전압/전력 데이터 분석
로봇 관절 제어	위치 계산, 속도 제어, 보간
시뮬레이션	가상 공정 제어, 예측 계산
데이터 로깅	파일 생성, 문자열 파싱, 평균 누적

✅ ST vs 다른 언어 비교

항목	LD	FBD	SFC	ST
시각적 표현	매우 높음	높음	높음	없음
복잡한 연산	낮음	중간	낮음	매우 높음
조건 처리	보통	보통	분기형	매우 뛰어남
반복/배열	불가	제한적	제한적	완전 지원
디버깅	쉬움	쉬움	쉬움	어려움(로그 필요)
가독성	전기인에 적합	보통	상태 위주	개발자에 적합
적용 영역	간단 제어	논리 흐름	시퀀스	고급 제어 로직

✅ 정리

- **ST는 고급 제어 로직과 복잡한 연산에 적합한 PLC 언어**
- **고급 프로그래머, 시스템 통합자, 자동화 설계자**에게 특히 유용
- **조건, 반복, 수학, 배열, 사용자 정의 함수**까지 다양한 고급 기능 지원
- 현장에서는 LD/FBD와 병용해 **핵심 연산 로직은 ST로 구현**하는 경우가 많다

4.7 언어 간 비교 및 변환 방법

✅ 개요

PLC 언어는 IEC 61131-3 표준에 따라 **5가지 주요 언어(LD, FBD, ST, SFC, IL)**가 정의되어 있다.
각 언어는 **표현 방식, 사용 목적, 가독성, 복잡도 처리 능력이 다르며**,
같은 제어 로직을 서로 다른 언어로 표현할 수 있다는 점이 특징이다.

📌 프로젝트 성격, 개발자의 숙련도, 장비 요구사항에 따라 **적절한 언어를 선택하거나 병행 사용**해야 한다.

✅ 1. 언어 간 주요 비교표

기준	LD (Ladder)	FBD (Function Block)	ST (Structured Text)	SFC (Sequential FC)	IL (Instruction List)
표현 방식	릴레이 회로식	블록 연결 흐름도	Pascal형 텍스트 코드	상태 전이도	어셈블리형 명령어
시각화	매우 좋음	좋음	없음	매우 좋음	없음
가독성	전기 기술자에 우수	논리 흐름 표현에 우수	개발자 친화적	공정 순서에 최적	유지보수 어려움

기준	LD (Ladder)	FBD (Function Block)	ST (Structured Text)	SFC (Sequential FC)	IL (Instruction List)
연산/수학	매우 제한적	보통	매우 강력	약함 (내부 Action 필요)	약함
반복문/배열	불가	불가	완전 지원	불가	불가
조건 분기	단순 분기만	단순 분기만	복합 분기 모두 가능	전이 조건으로 분리	Jump/Call 제한적
추천 분야	전기 자동화, 설비 유지보수	논리 제어, 간단 자동화	수치처리, 복합 알고리즘	공정 시퀀스, 로봇 순서	구형 PLC 유지보수

✅ 2. 언어 간 변환 가능성

모든 언어는 **논리적으로 동일한 결과를 만들어낼 수 있다**.
그러나 **표현력, 간결함, 구현 난이도**에서 차이가 발생하며, 일부는 수동 변환이 필요하다.

변환 방향	가능 여부	주의사항
LD → FBD	자동화 도구 가능	병렬 논리 연결 방식 유지 필요
FBD → LD	대부분 가능	블록 단위 해석 필요
LD/FBD → ST	가능 (중간 코드 해석 필요)	논리를 조건문으로 전개
ST → LD/FBD	제한적	복잡한 연산/반복은 변환 불가
ST → IL	가능	대부분 컴파일러 내부에서 처리됨
SFC ↔ LD/ST	병용 구조 (Step 내부 Action으로 연동)	직접 변환보다는 설계 목적에 따라 구분

✅ 3. 언어 변환 예시

📌 예제: X0 AND X1 → Y0

- **LD**

```
[ X0 ]---[ X1 ]---( Y0 )
```

- **FBD**

```
X0 ─┐
    ├─ AND ──> Y0
X1 ─┘
```

- **ST**

```
IF X0 AND X1 THEN
    Y0 := TRUE;
ELSE
    Y0 := FALSE;
END_IF;
```

- **IL**

```
LD X0
AND X1
OUT Y0
```

📌 예제: 10초 타이머 후 램프 ON

- **LD**

```
[ X0 ]──[ TON T1 10s ]──( Q )──( Y0 )
```

- **FBD**

```
X0 ──> TON (PT := T#10s) ──> Q ──> Y0
```

- **ST**

```
TON1(IN := X0, PT := T#10s);
Y0 := TON1.Q;
```

✅ 4. 실제 산업 현장에서의 혼합 사용 전략

목적	추천 언어 조합
전통 릴레이 제어	LD + FBD
복잡한 수학 계산	ST
다단계 공정 시퀀스	SFC + ST (Action)
학습용/디버깅	LD / FBD
재사용 가능한 기능 구현	Function Block (FB) + ST
프로젝트 템플릿 구성	LD로 전체 흐름 → ST로 연산 → SFC로 시퀀스 제어

✅ 5. 자동 변환 도구 및 IDE 지원 예

소프트웨어	언어 간 변환 지원
Siemens TIA Portal	LD ↔ FBD 자동 변환, ST 병용
Mitsubishi GX Works3	LD ↔ ST 제한적 변환
Codesys	LD, ST, FBD, SFC 자유 병용
Beckhoff TwinCAT	ST 중심 + 시각 언어 병용
Omron Sysmac Studio	FBD ↔ LD 변환, ST 부분 가능

✅ 정리

- **모든 IEC 61131-3 언어는 동일한 논리 구현이 가능하나**,
  표현 방식, 목적, 가독성, 효율성에서 차이가 있다.
- 간단한 제어는 LD/FBD, 고급 연산은 ST, 공정 흐름은 SFC로 구분해 사용하는 것이 일반적이다.
- **한 프로젝트 내에서 다양한 언어를 병용하며**,
  각 기능 목적에 따라 적절한 언어를 분리 구성하는 것이 모범 사례다.