# ST (Structured Text) Basic

## For

# OpCon programmer

**Rev. 01**

지은이: 고기동

# 차례

1. 소개
2. ST 문법
    2.1 Statement
    2.2 Expression
    2.3 대입
    2.4 FB 호출
    2.5 Return
    2.6 IF
    2.7 CASE
    2.8 LOOP
    2.9 EXIT
    2.10 ; (라인종료)
    2.11 주석
3. Data type
    3.1 Standard data type
        1)BOOL
        2)Integer type
        3)Real type
        4)String
        5)Time & Date type
    3.2 Defined data type
        1)배열(Array)
        2)포인터(pointer)
        3)구조체(Structure)
        4)열거형(Enumeration)
        5)참조형(Reference)
저작권

1. 소개

ST 는 Structured Text 의 약자로 IEC61131-3 표준 텍스트형 프로그래밍 언어이다.
이 ST 는 High-Level 언어로서 Low-Level 기계어 수준 명령어를 사용하지 않으며 매우 복잡한 기능을 쉽게 구현할 수 있게 해준다. 뿐만 아니라 ST 는 C 또는 PASCAL 과 매우 유사하기 때문에 이들 언어를 사용할 줄 안다면 약간의 문법적 규칙을 익히는 것 만으로 곧바로 ST Programming 을 할 수 있다.

Bosch 의 CPS-21, VPP-21, VEP-30, CPS-21L programming 을 하려면 바로 이 ST 라는 언어를 반드시 사용할 줄 알아야 한다. Control computer 는 날로 발전을 거듭하고 있으며 이에 따라 프로그래밍 언어도 계속 발전하고 있다 기존 재래식 PLC 에 사용되던 Ladder diagram 이나 IL 은 그 구현능력에 한계가 있기 때문에 복잡한 연산,통신, 데이터 처리 등을 요구하는 요즘의 장비를 콘트롤 하기 위해서는 이제 고급 언어화된 ST 언어를 사용해야 한다.

## 2. ST 문법

모든 언어에는 말을 하는 규칙 즉 문법이라는 것이 존재한다. 우리가 말을 함에 있어서도 한국어는 한국어 문법에 맞게 , 영어는 영문법에 맞게 말을 해야 서로간에 무리 없이 대화를 할 수 있듯이 Programming 을 하기 위해서는 Programming 언어의 문법을 익히고 그 문법에 맞게 Programming을 해야 한다.

그런데 Programming 언어의 문법은 인간 언어의 문법보다 무척 더 엄격하다는 특징이 있다. 우리가 말을 할 때에는 문법적으로 다소 틀리게 구사했다 하더라도 그 정도가 너무 심하지 않다면 의사소통에 별 문제가 없다. 그러나 Programming 언어는 그 문법 규칙에 조금이라도 어긋나면 바로 에러가 발생하고 실행코드를 생성할 수 없게 되기 때문에 문법을 반드시 지켜줘야 한다.

본서 에서는 ST를 사용하기 위한 기본 문법에 대해 다루도록 하겠다.

ST 의 문법은 앞서 말한 바와 같이 C 나 PASCAL 과 매우 유사한 형태를 갖추고 있다. C 언어는 가장 널리 알려진 언어로서 Programmer가 알아야 할 필수 언어라 해도 과언이 아니며 이에 관련된 교재가 시중에 많이 나와 있으니 구해서 공부하면 ST 언어를 사용 하는 데에도 많은 도움이 되리라 본다.

### 2.1 Statement

Statement 라 함은 우리 식으로 표현하자면 '문' 정도가 될 것이다. 저급 언어에서는 명령(instruction) 이라는 개념을 주로 사용한다.

그래서 흔히들

"덧셈연산을 하기 위해서는 add 라는 명령을 사용한다."

"jmp 명령은 분기 명령이다"

라는 식의 표현을 많이 사용한다. 그러나 고급언어로 넘어오게 되면 **명령어 라는 개념은 더 이상 사용되지 않는다.** 그리하여 고급언어 에서는 statement 즉 " xx문" 이라는 새로운 개념의 표현을 사용하게 되는데 저급언어에서의 명령어 와는 그 의미와 사용법이 매우 다르기 때문이다. 이 시간 이후 앞으로 여러분은 ST 뿐 아니라 C , C++ 와 같은 고급언어를 논할 때 어떤 "명령"을 사용하느니 어쩌니 하는 전혀 어울리지 않고 **무식한(?)** 표현을 사용하지 않길 바란다.

| Keyword | Description | Example | Explanation |
|---|---|---|---|
| := | 대입 | D:=10 | := 우측의 값을 좌측으로 대입 |
| | FB 호출 | FBName(<br>Par1:=10,<br>Par2:=20); | Function Block을 그 파라미터와 함께 호출 |
| RETURN | 리턴(복귀) | RETURN; | 현재 POU를 떠나 자신을 호출한 POU로 복귀 |
| IF | 조건적 선택 | IF d<e THEN f:=1;<br>ELSIF d=e THEN f:=2;<br>ELSE f:=3;<br>END_IF; | IF 와 THEN 사이의 조건이 참이면 이어지는 문을 실행하고 참이 아니면 ELSE 나 ELSIF 를 실행 |
| CASE | 다중선택 | CASE f OF<br>1: g:=11;<br>2: g:=12;<br>ELSE g:=FunName();<br>END_CASE; | f 의 값이 뭐냐에 따라 여러 개의 문장중 선택실행. |
| FOR | 반복(루프1) | FOR h:=1 TO 10 BY 2 DO<br>f[h/2]:= h;<br>END_FOR; | 시작 값으로부터 종료값에 도달할 때 까지 지정된 증가값만큼 증가시키면서 구문 반복 실행 |
| WHILE | 반복(루프2) | WHILE m>1 DO<br>n:=n/2;<br>END_WHILE; | 루프 시작부에 정의한 종료조건에 일치할 때까지 구문 반복 |
| REPEAT | 반복(루프3) | REPEAT<br>i:= i*j;<br>UNTIL i<10000<br>END_REPEAT; | 루프 끝부분에 정의된 종료 조건에 일치할 때 까지 구문 반복 |
| EXIT | 루프종료 | EXIT; | 루프문의 강제 종료 |
| ; | 구문 종료 | ;; | 한 구문의 종료를 나타냄. |

Table 1 [ST statements]

이상이 ST에서 사용하는 “문”이다.

ST에는 무조건 jump 구문인 GOTO 문이 없다. 이는 기술적 제한이 아니고 무조건 분기가 구조적 프로그래밍을 방해하기 때문에 포함시키지 않은 것이다. 모든 조건 분기는 IF 문에 의해 기술될 수 있다.

보다 상세한 사용법은 다음 장에서 다루도록 하겠다.

2.2 Expression : Partial statement in ST

Expression 은 문의 처리에 필요한 값을 만들어 낸다. 하나의 Expression 값은 각각의 Operand(피 연산자) 또는 Operand 간의 논리연산 결과에 의해 만들어진다.

다음은 Expression 의 예이다.

10 n/2 i<10000 FunName()

하나의 Expression 은 피연산자와 ST 연산자를 포함하고 있다. Expression의 기본적 구조는 다음과 같다.

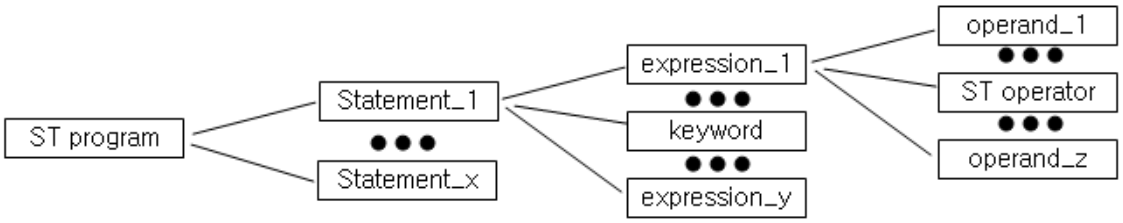


Figure 1 ST 프로그램은 문으로써 이루어져 있고 각 문은 키워드와 Expression으로 이루어져 있으며 각 Expression은 피 연산자와 ST 연산자로 이루어져 있다.

Operand(피 연산자)

Operand는 연산자에 의해 연산을 당하는 대상으로서 다음과 같은 종류가 있다.

- Literal (숫자,알파벳 문자,시간)

  Ex) 10, 'abcdef', t#3d_7h

- 변수(single/multi element variable)

  Ex) Val1, Var[1,2],Var1.Substruc

- 함수 호출

  Ex) FunName(Par1,Par2,Par3)

- 확장 Expression

  Ex) 10+20

Operator (연산자)

연산자는 수치연산자와 논리연산자가 있다.

수치연산자는 우리가 보통 사용하는 사칙연산(+-*/) 과 같이 수치 연산을 위한 연산자이며 논리연산자는 <>=,AND,OR 등과 같이 논리적 연산을 하기 위한 연산자이다. ST에서는 다음과 같은 연산자들을 제공한다.

| Operator | Description | Example & value of the expressions | Priority |
|---|---|---|---|
| ( ) | 괄호 | (2*3)+(4*5)<br>결과값:26 | High |
| | 함수호출 | CONCAT('AB','CD')<br>결과값: 'ABCD' | |
| ** | Exponentiation | 3 ** 4<br>결과값:81 | |
| - | 음수 | -10<br>결과값: -10 | |
| NOT | 반전 | NOT TRUE<br>결과값: FALSE | |
| * | 곱하기 | 10*20<br>결과값: 200 | |
| / | 나누기 | 20/10<br>결과값: 2 | |
| MOD | 나머지 | 17 MOD 10<br>결과값: 7 | |
| + | 더하기 | 1+2<br>결과값: 3 | |
| - | 빼기 | 2-1<br>결과값: 1 | |
| <,>,<=,<br>>= | 비교 | 10>20<br>결과값: FALSE | |
| = | 같다 | T#26h = T#1d2h<br>결과값: TRUE | |
| <> | 다르다 | 8#15<>13<br>결과값: FALSE | |
| &,AND | AND | TRUE AND FALSE<br>결과값: FALSE | |
| XOR | Exclusive OR | TRUE XOR FALSE<br>결과값:TRUE | |
| OR | OR | TRUE OR FALSE<br>결과값:TRUE | Low |

Table 2 ST 연산자와 연산 우선순위

2.3 대입

:= 를 사용하며 형식은 다음과 같다.(C 언어에서는 = 을 사용한다.)

Destination(variable) := source(expression);

즉 오른쪽에 있는 expression 의 결과값을 왼쪽에 있는 변수로 대입시킨다.

Ex)

```
VAR   a: INT;
      b: INT;
      c: INT;
END_VAR

a :=10;     (* a = 10*)
b := a *2; (* b = 10 * 2 =20   *)
c := b/a;   (* c = 20/10 = 2 *)
```

2.4 FB 호출

FB 호출은 다음과 같은 형식을 따른다.

<FB name>(FB 인자1, FB 인자 2.. FB 인자 n);

Ex)

CONCAT('AB','CD');

2.5 RETURN

RETURN 문은 Function, FB 또는 프로그램이 완전히 끝나지 않았더라도 벗어나고자 할 때 사용한다.

Ex)

```
   ....
(* 만약 x 가 y 보다 작다면 POU 를 빠져나가라 *)
  IF x < y THEN RETURN;
  END_IF;
```

2.6 IF 문

IF는 조건에 따른 선택을 하고자 할 때 사용되며 IF 문의 전형적인 형식은 다음과 같다.

```
IF   expression   THEN statement block;
ELS_IF expression THEN statement block;
                    .....
ELSE statement block;
END_IF;
```

IF 와 THEN 사이의 expression 결과값이 TRUE 이면 THEN 바로 다음의 프로그램 구문을 실행하며 결과 값이 FALSE 일 경우 그 다음의 ELS_IF 조건을 검사하고 그 결과 값이 TRUE 이면 THEN 바로 다음의 구문을 실행한다. ELS_IF 는 계속 연달아 사용할 수 있으며 맨 마지막에는 IF 와 ELS_IF 조건에 모두 해당하지 않을 경우 실행할 ELSE 이하 구문을 기술 해준 후 END_IF; 로써 IF 문을 마무리 짓는다.

CodeSys 에서 END_IF 다음에는 ; 을 넣어주지 않아도 별 문제 없다.

EX)

```
IF( var1 = 1 )
THEN
    Var2 := FALSE;
    Var3 := TRUE;
ELS_IF(var1 = 2 )
THEN
    Var2 := TRUE;
    Var3 := FALSE;
ELSE
    Var2 := FALSE;
    Var3 := FALSE;
END_IF;
```

**해석>** 만약 Var1 이 1 이면 Var2 를 FALSE, Var3 를 TRUE 로 만들고
그렇지 않고 만약 Var1 이 2 이면 Var2 를 TRUE, Var3를 FALSE로 만들며
이 두 가지 조건에 모두 해당되지 않으면 둘 다 FALSE 로 만든다.

## 2.7 CASE 문 (다중선택)

CASE 문은 다중 선택문으로서 하나의 expression 결과 값이 뭐냐에 따라 여러 개의 구문 중 하나를 선택하여 수행한다.

그 형식은 다음과 같다.

```
CASE expression OF
    Case_value1 : statement block;
    Case_value2 : statement block;
    Case_value3 : statement block;
    …..
ELSE statement block;
END_CASE;
```

Ex)

```
CASE   var1 OF
 1:     a:= 10;
 2:     a:= 30;
 3:     a:= 100;
 ELSE a:= 0;
END_CASE;
```

이 예제에서 Var1 의 값이 1 이면 a 에 10을 대입, 2 이면 30 대입,3 이면 100 을 대입하며 그 외의 값일 겨우 a 에 0 을 대입한다.

2.8 Loop (반복문)

Loop 는 어떠한 프로그램 루틴을 반복조건에 따라 반복시킨다.

ST에서 제공하는 Loop 는 다음 3가지 종류가 있다.

1) FOR Loop

Ex)

```
 FOR Idx:=1 TO 10
 DO
     Var1 := Var1 +1;
 END_FOR;
```

2) WHILE Loop

Ex)

```
 Idx := 10;
 WHILE Idx > 0
 DO
    Var1 := TRUE;
    Idx := Idx-1;
 END_WHILE
```

3) REPEAT Loop

Ex)

```
REPEAT
   i:= i*j;
   UNTIL i<10000
END_REPEAT;
```

2.9 EXIT

EXIT 는 Loop 문으로부터 빠져 나오고자 할 때 사용된다.

2.10 ; (라인 종료)

; 은 expression 의 끝을 나타내 준다.

어떠한 expression 종단에 ; 를 넣는 것을 빠트리면 컴파일시 에러가 발생한다.

Ex)

A := b+c; (* OK *)

A := b+c; E := a+d; (* OK 2개의 expresión 실행 *)

A := b+c E := a+d; (* error 발생*)

2.11 (* *) (주석)

Program 에 주석문을 넣고자 할때는 (* *) 를 사용한다.

Ex)

A := b+c; (* This is a sample comment *)

(* *) 로 감싸진 부분은 프로그램 컴파일시 무시된다.

(* 가 일단 나오면 *) 가 등장할 때 까지는 모두 주석으로 인식됨을 유의하기 바란다.

참고로 C 언어 에서는 /* */ 와 // 로 주석문을 처리함.

3. Data type

3.1 Standard Data type

1)BOOL:이진 1비트 type으로서 TRUE 와 FALSE 즉 1과 0 두 가지 값만 가질 수 있음.

2)Integer type : 정수 형태로서 다음과 같은 종류가 있다.

| Type | Lower limit | Upper limit | Memory space |
|---|---|---|---|
| **BYTE** | 0 | 255 | 8 Bit |
| **WORD** | 0 | 65535 | 16 Bit |
| **DWORD** | 0 | 4294967295 | 32 Bit |
| **SINT** | -128 | 127 | 8 Bit |
| **USINT** | 0 | 255 | 8 Bit |
| **INT** | -32768 | 32767 | 16 Bit |
| **UINT** | 0 | 65535 | 16 Bit |
| **DINT** | -2147483648 | 2147483647 | 32 Bit |
| **UDINT** | 0 | 4294967295 | 32 Bit |

3)REAL type: floating point type 이라고도 불리며 소수점 있는 실수형태의 값을 가질 수 있고 32 비트 크기의 REAL 과 64 비트 크기의 LREAL 2종류가 있다.

4)STRING(문자열):말 그대로 문자열로서 기본 문자열 사이즈는 80 문자이고 조정이 가능하다.

Ex)

STRING(35) := 'This is a String';

5)Time & Date type: 시간과 날짜를 값으로 갖는 Data type 이다.

TIME constants: Time 값을 갖는 type 이며 "t" 또는 "T" 와 "#" 를 써준 후 그 뒤에 시간 수치와 단위를 써준다.

시간 단위의 종류는 다음과 같다.

| ID | meaning |
|---|---|
| d | day |
| h | hour |
| m | minutes |
| s | sec |
| ms | mili-sec |

Ex1) 옳은 사용 예

TIME1 := T#14ms;

TIME1 := T#100S12ms; (*최상위 단위는 제한 값을 초과해도 된다 *)

TIME1 := t#12h34m15s;

Ex2) 잘못된 사용 예

TIME1 := t#5m68s; (* 하위 단위가 제한 값 초과*)

TIME1 := 15ms; (*T# 누락*)

TIME1 := t#4ms13d; (* 단위 순서 잘못됨*)

DATE Constants: "d", "D", "DATE" 또는 "date" 로 시작되고 "#" 다음에 Year-Month-Day 포맷의 날짜를 입력해 사용한다.

Ex)

DATE#1996-05-06

d#1972-03-29

TIME_OF_DAY Constants, for storing times of the day: "tod#", "TOD#", "TIME_OF_DAY#" 또는 "time_of_day#" 로 시작되며 Hour:Minute:Second 포맷의 값을 입력한다.

Ex)

TIME_OF_DAY#15:36:30.123

tod#00:00:00

DATE_AND_TIME Constants, combination of date and the time of day:
"dt#", "DT#", "DATE_AND_TIME#" 또는 "date_and_time#" 로 시작되며 년,월,일, 및 시간 사이는 hyphen 을 넣어준다.

Ex)

DATE_AND_TIME#1996-05-06-15:36:30

dt#1972-03-29-00:00:00

3.2 Defined Data type

1)배열(Array)

ST 에서는 1차원, 2차원 및 3차원 배열을 지원하며 기본 선언형식은 다음과 같다.

<Field_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>

ll1, ll2, ll3 은 field 범위의 최저 제한 값을 나타내며, ul1, ul2, ul3 는 filed 범위의 최대 제한 값을 나타낸다. 범위 값은 반드시 integer type 이어야 한다.

Ex)

Card_game: ARRAY [1..13, 1..4] OF INT;

배열의 초기화:
다음은 완벽한 배열선언의 예이다.

```
arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7);     (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3; (* short for 0,0,4,4,4,4,2,3 *)
```

구조체 배열의 선언 예:

```
TYPE STRUCT1
STRUCT
p1:int;
p2:int;
p3:dword;
END_STRUCT
ARRAY[1..3] OF STRUCT1:= (p1:=1;p2:=10;p3:=4723),(p1:=2;p2:=0;p3:=299),
(p1:=14;p2:=5;p3:=112);
```

배열의 부분 선언 예:

```
arr1 : ARRAY [1..10] OF INT := 1,2;
```

초기값을 지정하지 않은 배열 element 는 default 값 0 으로 초기화 된다.

2차원 배열의 각 component 는 다음과 같은 형식으로 access 한다.

<Field_Name>[Index1,Index2]

Ex)

```
Card_game [9,2]
```

2) 포인터(Pointer)

포인터(pointer)는 데이터가 기억되어 있는 기억장소의 주소(address)를 기호 주소 형태로 나타낸 것을 의미하며 ST 도 pointer를 제공한다.

포인터의 선언은 다음과 같은 형식을 따른다.

<Identifier>: POINTER TO <Datatype/Functionblock>;

포인터는 어떠한 data type 또는 function block 뿐 아니라 사용자 정의 type 도 가리킬 수 있다. 주소 연산자 ADR 은 변수나 Function block 의 주소를 포인터로 어사인 시킨다. 포인터는 포인터 identifier 다음에 content 연산자 "^" 를 추가하여 참조 시킨다.

Ex)

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
pt := ADR(var_int1);
var_int2:= pt^; (* var_int2 의 값은 5 가 된다 *)
```

3)구조체(Structure)

구조체(Structure)는 C 언어를 비롯한 각종 고급 프로그래밍 언어에서 사용하는 데이터 추상화의 중요한 기법 중 하나로, 하나 이상의 데이터 형을 가진 변수들의 모임을 의미한다.

본서는 기본적으로 C 언어와 같은 고급언어에 대한 기본 지식이 있는 프로그래머를 대상으로 하고 있기 때문에 구조체에 대한 자세한 설명은 하지 않기로 한다. 만약 구조체를 잘 모르는 독자라면 C언어 교습서 등과 같은 책을 구하여 공부하기 바란다.

구조체는 레지스터 카드 data type 하의 Object Organizer 안에 있는 object 로서 생성된다. 구조체는 TYPE 과 STRUCT 키워드와 함께 시작되며 END_STRUCT 와 END_TYPE 으로 끝맺는다.

구조체 정의 형식:

```
TYPE <Structurename>:
STRUCT
    <Declaration of Variables 1>
    .
    .
    <Declaration of Variables n>
END_STRUCT
END_TYPE
```

<Structurename>은 Project 내에서 인식되는 type 으로서 standard data type과 동일하게 사용될 수 있다.

구조체 연동은 허용된다. 유일한 제한 사항은 변수가 주소에 위치할 수 없다는 것이다. (AT 선언 불가).

Polygonline 이란 이름의 구조체 선언 예:

```
TYPE Polygonline:
STRUCT
    Start:ARRAY [1..2] OF INT;
    Point1:ARRAY [1..2] OF INT;
    Point2:ARRAY [1..2] OF INT;
    Point3:ARRAY [1..2] OF INT;
```

```
    Point4:ARRAY [1..2] OF INT;
    End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

구조체 초기화 예:

```
Poly_1:polygonline := ( Start:=3,3, Point1 =5,2, Point2:=7,3, Point3:=8,5, Point4:=5,7, End := 3,5);
```

변수와 함께 초기화 하는 것은 불가능 하다. 구조체 배열 초기화는 '배열' 단원의 구조체 배열 항목을 참조하기 바란다.

구조체는 다음과 이 **".**" 을 사용해 access 할 수 있다:

<Structure_Name>.<Componentname>

예를 들어 여러분이 "Monday"라는 component를 갖고 있는 "Week"라는 구조체에서 Monday를 사용 하고자 한다면 다음과 같이 하면 된다.

Week.Monday

4)열거형(Enumeration)

열거형(Enumeration)은 데이터가 가질 수 있는 모든 값들을 나열함으로써 정의되는 데이터 들의 집합 구조를 의미한다. 열거형의 값은 POU 내에서 선언된 것들까지도 project의 모든 영역에서 인식 된다.

레지스터 카드 data type하의 object organizar 의 object로서 열거형을 생성하는 것이 가장 좋다. 열거형은 TYPE 키워드로 시작되고 END_TYPE 으로 끝맺는다.

Cf.) C언어 에서는 enum 이라는 별도의 키워드를 사용함.

Syntax:

```
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);
END_TYPE
```

<Identifier>란 이름의 변수는 열거형 값을 갖고 Enum_0 부터 Enum_n 까지의 원소를 갖는다. 원소의 값은 INT type 이며 각 원소에 대해 임의의 순서를 지정하지 않는 한 순서대로 1씩 증가하며 값을 할당한다. 만약 첫 번째 원소가 초기화 되지 않으면 자동으로 0부터 시작된다.

Ex)

```
TYPE TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (*각 color의 초기값은
                                                red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* traffic signal 의 값은 red*)
FOR i:= Red TO Green DO
    i := i + 1;
END_FOR;
```

같은 열거형 값은 두 번 사용될 수 없다.

Ex)

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
```

Error: red 는 TRAFFIC_SIGNAL 과 COLOR 양쪽에서 사용되면 안된다.

5)참조형(Reference)

ST 에서는 사용자 정의 참조형 data type 을 다양한 이름으로 생성하여 변수,상수 또는 Function Block 에서 사용할 수 있다. 즉 새로운 데이터 형을 정의하여 사용하는 것이다. 참조형도 레지스터 카드 data type 하의 Object Organizer 안에 있는 object 로서 생성 된다. 형식은 키워드 TYPE 으로 시작해 END_TYPE 으로 마무리 한다.

Cf.) C언어에서는 typedef 라는 키워드를 사용한다.

Syntax:

```
TYPE <Identifier>: <Assignment term>;
END_TYPE
```

Example:

```
TYPE message:STRING[50];
END_TYPE;
```

4. 맺음말

이상으로 간단하게 나마 ST Language 의 사용법에 대해 알아 보았다. 다른 프로그래밍 언어를 접해본 독자들은 ST 가 C언어와 같은 여타 고급언어와 별반 다르지 않다는 것을 느꼈을 것이다. 즉, 약간씩 각 언어마다의 차이점은 있으나 대부분의 컴퓨터 언어란 것들은 대동소이한 것이다. 하나만 제대로 알면 다른 것들도 큰 어려움 없이 익혀 사용할 수 있다. 필자도 처음에 특별히 ST 언어를 배우지 않은 상태에서 약간의 스터디 만으로 바로 실전에 들어 갔었다. 따라서 ST를 잘 사용하기 위해선 우선 C언어를 충분히 공부 하기를 권하는 바이다. 모쪼록 이 글이 독자들에게 조금이나마 도움이 되기를 바라는 바이다.

Copyright
본 서의 저작권은 고기동 에게 있으며 무단전재 및 무단복제를 금함.